Kriging-based Self-adaptive Cloud Controllers

Alessio Gambi, Mauro Pezzè, Senior Member, IEEE, and Giovanni Toffetti

Abstract—Cloud technology is rapidly substituting classic computing solutions, and challenges the community with new problems. In this paper we focus on controllers for cloud application elasticity, and propose a novel solution for self-adaptive cloud controllers based on Kriging models. Cloud controllers are application specific schedulers that allocate resources to applications running in the cloud, aiming to meet the quality of service requirements while optimizing the execution costs. General-purpose cloud resource schedulers provide sub-optimal solutions to the problem with respect to application-specific solutions that we call cloud controllers. In this paper we discuss a general way to design self-adaptive cloud controllers based on Kriging models. We present Kriging models, and show how they can be used for building efficient controllers thanks to their unique characteristics. We report experimental data that confirm the suitability of Kriging models to support efficient cloud control and open the way to the development of a new generation of cloud controllers.

Index Terms—Self-adaptive controllers, cloud, IaaS, Kriging models

1 INTRODUCTION

Cloud technologies are spreading so quickly that soon “cloud-based solutions will be growing at a faster rate than on-premises solutions” [1]. According to Forrester Research, cloud is forecast to continue as a fundamental computing environment, and the cloud industry will grow from $40.7 billion in 2011 to $241 billion in 2020 [2].

The success of cloud technologies is driven by the convenience for both the infrastructure and the application providers. The virtualized approach that characterizes the cloud simplifies the resource management for the infrastructure providers enabling effective multi-tenancy support. The pay-per-use model enables application providers to reduce the running costs by leveraging dynamic resource allocation through elastic applications [3].

In this paper we target the Infrastructure as a Service (IaaS) paradigm that offers computing resources to application providers as an on-line service, on-demand and through a pay-per-use model. In the IaaS paradigm the application providers are directly responsible for deciding the amount of resources allocated to their systems, they must decide for example how much computing power, memory and disk resources the system requires, and how these allocations should evolve over time.

The choice of assigning the responsibility of allocating the required resources to the application developers depends on the nature of the cloud-based applications that typically include several heterogeneous components implementing the application logic through complex application-specific interactions. Hence, general-purpose cloud resource schedulers, like the ones monitoring only low-level system

1. Gambi is with the Vienna University of Technology, Vienna, Austria. E-mail: alessio.gambi@tuwien.ac.at
2. Pezzè is with the University of Milano Bicocca, Italy, and at the University of Lugano, Switzerland. E-mail: mauro.pezze@usi.ch
3. Toffetti is with the IBM Haifa Research Laboratories, Israel. E-mail: giovanni@il.ibm.com

Manuscript received XXXX XX, XXX; revised XXXX 17, 2014.

metrics, such as the virtual machines CPU usage, would be sub-optimal with respect to application-specific solutions that we call cloud controllers.

Cloud controllers must solve application-specific resource allocation problems in presence of two conflicting requirements. Application providers aim to offer an agreed Quality of Service (QoS) that improves with an increasing amount of resources assigned to the application, and at the same time would like to minimize operating costs that reduce with a decreasing amount of allocated resources. The goal is therefore striking the correct application scaling avoiding both under- and over-provisioning in presence of a fluctuating input workload.

The current state of practice for dynamically controlling the scale of cloud applications relies on threshold-based rules, which own their popularity to their simplicity and intuitiveness, and are offered by many commercial cloud providers such as Amazon1 and RightScale.2 The common practice is to define a small3 and readable set of scaling rules, assuming a linear and stable dependency between resource assignments and QoS improvements, while in practice, the complexity of component interactions, the interferences among component and the frequency by which hardware and software contention issues arise in cloud systems typically invalidate these assumptions [4], [5].

The research community has investigated many alternative approaches. Solutions based on control theory rely on similar assumptions of rule-based controllers and thus suffer from analogous limitations; classic approaches to capacity planning based on queuing theory or similar analytic models do not fully address the dynamics of cloud applications; the recent trends towards the definition of self-adaptive controllers seem to adapt better to the complexity of cloud controllers [6].

3. Infrastructure providers typically limit the amount of definable rules for their automated solutions.
In this paper we contribute to advance the knowledge in self-adaptive controllers by presenting a model-driven approach to automatically engineer application-specific cloud controllers. We propose Kriging models [7] to approximate the complex and a-priori unknown relations between (1) the non-functional system properties collected with runtime monitors, like responsiveness, availability and throughput, (2) the system configuration, for example the amount of virtual machines, and (3) the service environmental conditions, for instance workload mix, workload intensity and interferences.

Kriging models offer several advantages over other modeling approaches: They effectively deal with multi-dimensional configuration spaces, support fast update up to a considerable number of samples enabling runtime training, and offer accurate predictions that consistently improve with the size of the available system data.

While various families of models are comparable to Kriging models in some of the above aspects, what makes Kriging models unique is that each prediction they provide comes with a confidence measure [8] that can be effectively used to implement several appropriate reaction policies in case of model uncertainty, therefore leading to more robust solutions.

This article extends our previous work where we propose Kriging models to capture the performance of cloud-based systems [9] and discuss the viability of Kriging-based controllers [10] in several ways:

1) We introduce novel modeling solutions, namely Kriging+, that exploit the unique features of Kriging models to improve performance and robustness of cloud controllers.
2) We present a prototype implementation of the controllers and the tools supporting their automatic configuration that confirms the viability of our solution and that we used for experimenting with the approach.
3) We show the validity of the approach by means of an extensive set of experimental data collected on an industrial case study.

The rest of this paper is organized as follows. In Section 2, we introduce the problem of elastic control for the cloud. In Section 3, we present Kriging and Kriging+ models focusing on the aspects relevant for the paper. In Section 4, we show how Kriging models can be effectively used as the core component of autonomic cloud controllers and describe the architecture and the internals of Kriging-based controllers. In Section 5, we illustrate the experimental results and discuss the advantages and limitations of Kriging based cloud controllers. In Section 6 we survey the current state of the art and practice. In Section 7 we illustrate the main results presented in the paper.

2 ELASTIC CONTROL FOR THE CLOUD

The cloud paradigm decouples software applications from their execution environment by introducing a stack of abstraction layers that isolate the computation at different levels: the execution infrastructure -Infrastructure as a Service (IaaS)- the overall platform and runtime environment -Platform as a Service (PaaS)- and the provided services -Software as a Service (SaaS)- [11]. In this paper, we focus on the IaaS layer that takes care of allocating resources to applications.

In a general IaaS scenario, application providers run their applications in the form of virtual machines (VMs) that are executed in the IaaS cloud hardware infrastructure. Typically applications are composed of several types of virtual machines each providing some specific services. The services offered by the virtualized applications are accessed by the end-users who generate workload that may vary unpredictably over time.

Applications try to acquire as many resources as possible to guarantee the service qualities specified by their service level agreements (SLA), which usually associate penalties to SLA violations, while the application providers try to minimize the allocated resources to reduce costs and increase competitiveness and profit [12].

Application providers monitor the level of service of the applications through a monitoring interface that provides high level application-specific metrics also referred to as Key Performance Indicators (KPIs). Application providers can use the control interface of the IaaS to manage the number and type of running VMs as well as the assigned physical resources for their service.

The elastic control problem is complicated by many factors. The workload of cloud applications changes over time both in terms of intensity (request rates) and types of requests (workload mix). While periodic workload fluctuations can be identified and modeled, it is typically hard to predict workload mix and request rates with high degree of precision, especially in presence of extreme peak situations such as slash dot effects and flash crowds that characterize many cloud applications [13].

The variety of resources with different characteristics and costs offered by IaaS providers and the complexity of the applications, the intertwining of their components and their specific interactions make it very hard to estimate the effects that any reconfiguration action will have on the running system. Some system reconfiguration actions, for instance starting or migrating a VM, are not immediate and the complete actuation might require waiting time in the order of minutes, thus further complicating the prediction of the expected behavior from the data that are currently monitored.

The problem of building efficient elastic controllers is targeted in many ways that differ both in the models used to capture the knowledge of the system and in the strategies proposed to identify a suitable configuration while reacting to changes in the workload conditions.

As discussed in Section 6, we identify three main categories of controllers: threshold- and rule-based, control-theoretic and self-adaptive (model-based) controllers. In this paper, we propose a new type of black-box self-adaptive controller based on Kriging models that promises to address well all the problems of elastic control.

3 KRIGING MODELS FOR SELF-ADAPTIVE CONTROL

In this section we introduce Kriging and Kriging+ models that we propose as the core component of self-adaptive controllers. To make the paper self-contained we summarize
how Kriging models work in line with the discussion originally presented by Jones et al. [14]. The interested reader can find a comprehensive presentation of Kriging models in [15, 16].

Kriging models are a family of black-box models that approximate unknown functions via interpolation and regression [17]. We propose Kriging models to capture the relations between the measurable behavior of cloud applications and their system configuration, input workload, and possibly other metrics. For example, we use Kriging models to capture systems performance as a function of the amount of allocated resources, and intensity and mix of the input requests. We design controllers that exploit Kriging models for planning their control, and that continuously adapt to the ever-changing dynamics of the cloud, thus leading to self-adaptive control.

Kriging models have many interesting properties that make them suitable for our purpose: (i) They are global, which means that they span over the whole input space, differently form classic regression models that are suited for local predictions only; (ii) they can deal with multi-variate functions; (iii) they can capture linear, non-linear and multi-modal relations between the data as long as the unknown function is smooth and continuous; (iv) they are non-parametric, meaning that no predefined structure is imposed on the model, differently from traditional parametric models, such as polynomial regressors; (v) they quantify the uncertainty of the predictions accounting for the availability of training samples, their distribution over the input space and their level of noise; (vi) they do not need a large amount of training samples to provide acceptable predictions, and thus are relatively fast to train.

Kriging models extend traditional regression models with a statistical framework based on stochastic processes, but shift the original emphasis of traditional regression from the regression terms to the error terms. Therefore, instead of estimating the regression coefficients that together with an assumed polynomial describe what the function is about, Kriging models estimate the correlation parameters that describe how the function typically behaves.

To do so, Kriging models assume a spatial correlation between observations, that is, observations close one to each other in the input space should have similar output values. Kriging models capture this correlation by means of a correlation matrix of a stochastic process. They predict the value of the unknown function in a test location (denoted by $x_*$) according to the best linear unbiased predictor (1), and estimate the accuracy of predictions as the mean squared error (2).

$$\hat{y}(x_*) = \mu(x_*) + k^T K^{-1} (y - \mu(x_*)) 1$$  (1)

$$s^2(x_*) = \sigma^2 [1 - k^T K^{-1} k + \frac{(1 - k^T K^{-1} k)^2}{k^T K^{-1} k}]$$  (2)

In (1) and (2), $\mu(x_*)$ is the value of the (optional) regressor terms in the test location, $K$ is the correlation matrix, which does not depend on the input location, $y$ contains the values of the collected observations, $k$ is the vector of correlations between the test location and the collected observations, and $\sigma$ is the standard deviation of the error terms.

We can interpret the first term of $\hat{y}(x_*)$ as the information provided by the assumed model, and the second term as the adjustment based on the correlation among the errors. We can interpret the term $(k^T K^{-1} k)$ in $s^2(x_*)$ as the reduction in prediction error due to the correlation of the test location with the observations, and the remaining term as the uncertainty that stems from not knowing exactly the unknown function.

Both $\hat{y}(x_*)$ and $s^2(x_*)$ correctly reflect the intuition behind the spatial correlation: When $x_*$ is far from the observed data there is no correlation ($k = 0$), therefore the predictor reduces to the assumed model and there is no adjustment to the mean square error; when $x_*$ corresponds to a previously observed location the correlation is maximal ($k = K$), therefore the predictor interpolates the data exactly; between these extremes the values of the predictor and the square mean error are derived as a combination of the values of the training observations weighted by their distance from the test location.

We use the standard distance metric for Kriging models that results in the correlation elements of (3), where $(h)$ identifies the index of input dimensions.

$$K_{i,j} = \exp[- \sum_{h=1}^{H} \theta^{(h)} |x_i^{(h)} - x_j^{(h)}|^p]$$  (3)

The behavior of this distance metric is governed by a set of hyper-parameters that determine for each input dimension the smoothness of the function ($\rho$) and its activity ($\theta$). Referring to this set of parameters we can associate different weights with different input dimensions, thus capturing their relative importance.

Since we use Kriging models to describe real systems, we cannot assume that observations are noise-free. To account for noise in the data, we follow the approach presented by Forrester et al. [16], and we introduce an additive “noise” parameter in the diagonal of the correlation matrix. Assuming a Gaussian noise, all the previous conclusions derived from Kriging models remain valid.

Adding the noise term produces two effects: The training process must estimate additional parameters, and the resulting model implements a form of data smoothing; therefore, the Kriging model might not perfectly interpolate the data.

In the remaining of this paper, even if not explicitly mentioned, we refer to this noise augmented formulation of Kriging models.

**Kriging+: combining models to deal with uncertainty**

Usually, the training set contains samples that are scattered across the input space, and that form clusters. Kriging models provide accurate predictions within and nearby those clusters where the density of observation is high, but less accurate predictions in regions where data are scarce or absent. In these regions, analytical models, such as queuing networks, whose accuracy does not depend on the amount and distribution of observations, might provide more accurate predictions than Kriging. To improve the overall prediction accuracy, analytical models can be used...
as “fallback mechanisms” for Kriging models when Kriging models may provide low accuracy predictions.

Kriging+ models implement this concept by leveraging the ability of Kriging models to quantify the uncertainty of their predictions for deciding which one of the “internal” models is the most reliable, and then uses that model to compute the actual prediction. In particular, given the target location \( x \), a Kriging+ model computes the Kriging value \( s^2(x_*) \) (according to Equation 2). If the computed value is below a user-configurable threshold, Kriging+ considers the Kriging model accurate enough for the prediction, otherwise it uses an alternative analytical model.

4 Kriging-based Controllers

In this paper, we claim that the characteristics of Kriging models, and, in particular, the ability of dealing with multi-dimensional configuration spaces, the possibility of fast update and query and the estimation of the accuracy of the predictions, make them suitable for supporting the control of complex systems. In this section, we discuss how Kriging models can be used to build efficient cloud controllers, and present a high level characterization of the process needed to build a Kriging-based controller starting from an application and some control goals, all the way to a working controller. From this process we derive a list of requirements for our controller implementation, and in the remaining of the paper, we show how Kriging-based controllers meet well these requirements.

In the reference IaaS settings, the controlled system is composed of virtualized applications that run on top of the physical hardware and network infrastructure provided by the cloud. The system exposes a public service interface and a management interface. The public service interface is used by the end-users of the application, while the management interface enables the controller both to monitor the system configuration and performance, and to apply control actions. In operational conditions, the system is subject to a varying input workload that can be characterized according to different dimensions, for instance, average inter-arrival time of request per type, average request size, workload mix. We assume that the controller does not limit the admission to the services and thus the workload cannot be altered by the controller.

A generic controller for cloud applications is characterized by the following dimensions:

- **Workload metrics** \( \overline{W} \) that describe the workload intensity and composition, and depend on the elements monitored by the application. For instance, a Web application could measure the number of invocations for each served URI over a period of time (arrival rate per operation), the workload intensity including all request types, or both, as well as many other elements.

- **Service configuration** \( \overline{SC} \) that is a superset of the entities and attributes modifiable through control actions. For example, it may include the number of running instances for each VM type of the application, their virtual resource assignment, as well as other application-specific tunable parameters (for instance, thread pool sizes, heap size, etc).

- **Performance metrics** \( \overline{P} \) that quantify the application performance. Depending on both the infrastructural and application-specific monitoring systems, several performance metrics might be available for an application, for instance throughput and response time of each operation. Application providers might also expose internal application metrics revealing additional insight on the system internal state or behavior, in the form of key performance indicators.

**Control goals** \( G \) that combine workload, service configuration and performance metrics into a specification of desired system behavior. Control goals can be specified in many ways, in this work we assume that control goals are represented as an optimization problem, that is, the maximization of a configurable utility function over the controlled period. We only require that the utility function is expressed exclusively in terms of the metrics available to the controller: the workload metrics, the service configuration, and the system performance response metrics as in Equation 4.

\[
G = f(\overline{W}, \overline{SC}, \overline{P})
\]

A typical control goal for cloud applications is the maximization of the application revenue that can be represented as a function of the number of served requests, the responses that do not meet the required QoS and the cost of running the VMs.

A Kriging-based controller includes a Kriging model for each performance metric \( p \) included in \( G \) (Equation 5), and combines the predictions of all the models to compute the expected utility at each control step.

\[
\forall p \in \overline{P}, \ p = \hat{g}(\overline{W}, \overline{SC})
\]
C. GOAL

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2015.2389236, IEEE Transactions on Services Computing

documents that specify the resource requirements of the virtual machines and networks, the KPI and the performance objectives (SLO), and configures the remaining components. The Monitoring connector and the Actuator are extensible components that connect the controller to the controlled system. The monitoring connector periodically collects data about service configuration and performance indicators, and stores them in the KPI database, while the actuator maps control policies into commands for specific IaaS platforms and application-specific reconfiguration actions. Our current implementation provides interfaces for Amazon EC2, OpenStack and the Reservoir$^5$ platform. The Model manager builds and keeps up-to-date the models used by the controller, and provides an interface for requesting model predictions. The initial training and the following updates of the models are based on historical data that the model manager elaborates according to configurable update policies. Our default implementation manages Kriging models, and preprocesses the training data to reduce noise by averaging performance measures over a configurable number of samples. The Configuration selector is the heart of the controller and implements its control logic. The configuration selector combines information about the service configuration, the performance goals, the current monitoring information and the predictions of employed models to implement user provided control policies. Our default control policy aims to find the smallest (defined as the cheapest) system configuration that satisfies all the performance objectives provided in the service manifest.

Controller implementation

We built a Java prototype implementation of the architecture shown in Figure 1. Each component has been developed as a Maven module to simplify and manage library dependencies. The parser, the model manager and the configuration selector are cloud platform-independent components. The model manager uses the Java Octave package to interface with the Java programming language. We leverage the oct-gpr extension provided by Octave for building Kriging models (or Gaussian process regression, hence “gpr”).

The monitoring connector and the actuator components interact directly with the cloud for gathering performance data, and for retrieving and changing system configurations, for instance, by adding or removing VMs; therefore these components are cloud-specific. We provide implementations of the monitoring connector and actuator for the Reservoir, the OpenStack and the Amazon EC2 platforms: The Reservoir implementation uses the libraries for accessing the Claudia service manager interface, Apache jclouds is used for OpenStack, while Typica is our library of choice for Amazon EC2.

Finally, some applications might expose their own specific performance KPIs and/or might require application-specific reconfiguration actions when VMs are added or removed from the system. To cater for these needs, we implemented configurable monitoring collectors that either listen to JMS queues or directly poll cloud-application monitoring databases. Similarly, for controlled applications we implemented programming hooks that allow VMs to (de)register when removed or added to an existing application.

5 Experimental Evaluation

In this section we report the data of the experimental evaluation of the approach. We evaluated the approach according to two main dimensions, the suitability of Kriging and Kriging+ models to serve as models for cloud applications and the possibility of building cloud controllers based on these models.

Below we discuss the experimental settings and the results that confirm our main research hypothesis:

Kriging and Kriging+ models can be used as the core element of efficient and effective self-adaptive model-based controllers.

We show that Kriging models and the other state-of-the-art models perform comparably good, with no clear winner across all the major points of our investigation. We also show that Kriging+ models over-perform all the other models in almost all the experiments. Furthermore, during our evaluation controllers based on Kriging+ models were the only ones that suitably balance the trade-off between QoS and costs in presence of realistic working conditions. Table 5 at page 11 summarizes in a qualitative way our findings and highlights the benefits of adopting Kriging+ models in the context of cloud-based control.

To answer the main research question we first evaluated how accurately Kriging and Kriging+ models capture the behavior of applications running in the cloud, in particular referring to the reliability of these models to predict the system behavior and thus the possibility of building truly effective controllers. Then we evaluated the other aspects of the models that are relevant in the development of model-based controllers.

We articulated these issues in three detailed research questions that are listed below:

**RQ1** How accurately do Kriging and Kriging+ models capture the behavior of cloud applications?

**RQ2** Are Kriging and Kriging+ models prompt enough to fit the self-adaptive control loop without impairing it?

**RQ3** Can Kriging and Kriging+ models adapt to the behavior of running applications by using the monitored data?

In the following, we describe the experiments and discuss the results that we obtained for each detailed research question and for the main research hypothesis. We contextualize our findings by comparing the results obtained with Kriging and Kriging+ models against the results obtained with competitive state-of-the-art solutions.

### 5.1 Experimental setting

In this section, we briefly describe the testbed infrastructure and the case studies that we used in the experiments.

We chose the Reservoir cloud as middleware for IaaS. This choice is motivated by our need to produce repeatable results experimenting with an industrial system, hence the availability of a private physical infrastructure at hand was a key requirement. In addition, our choice allowed us to leverage the experience that we gained by working in the Reservoir EU project⁶. We installed the Reservoir management components in a virtual machine running on a general purpose desktop, and we used a 24-core physical server to run up twenty-two single-vCPU user provided virtual machines.

During the evaluation, we used as case study representative systems that are typically executed in cloud infrastructures. These systems are a batch job processing system (*Sun Grid Engine*⁷), a REST Web service (a clone of *Doodle*⁸), a Web application ([RUBiS]⁹), and a composition of RESTful Web services ([DoReMap]⁸). The experiments on the different case studies provided similar results. Here we present extensive results only for the Sun Grid Engine (SGE).

The SGE has a master-slave architecture: clients submit jobs to the master node that first queues the jobs and then dispatches them to executor nodes. In our settings, the master runs in a 2-vCPU virtual machine. Each executor runs in a single-vCPU virtual machine, and can execute one job at a time. Executors can be dynamically added and removed, thus they form an elastic execution layer.

As for most current applications, the SLA for the SGE refers to the response time, and defines a threshold of 120 seconds on the average jobs completion time measured over a five-second sliding window. To create synthetic workloads we used homogenous jobs that in average complete in five seconds on the average jobs completion time measured over a five-second sliding window. We must notice that in presence of intense fluctuations in the load, the first with a peak request rate of 150 jobs/minute, and the second with a peak request rate of 55 jobs/minute; a ninety-five minute long workload that increases the average job rate stepwise up to 220 jobs/minute; a hundred-and-twenty minute long workload that slowly varies the average request rate between 10 and 70 jobs/minute (Profile 6, Profile 3, Profile 4, in the experiments, respectively).

### 5.2 Models for Self-Adaptive Control

We compare the results obtained with Kriging and Kriging+ models with the main alternative modeling approaches:

**Multidimensional Linear Regression (MLR)** models that take the form of \( y = X \cdot b + c \), and are usually trained via least square error minimization. In the evaluation, we use the implementation provided by the statistics package of [Octave]¹⁰.

**M5 Regression Trees (M5RT)** that correspond to the widely known classification trees in the continuous case, and are trained via a two-phase procedure that grows an initial set of each branch by means of standard deviation reduction, and then minimizes the tree by pruning each leaf that minimizes the estimation of the expected error. In the evaluation, we use the implementation of M5 regression trees provided by [Jekabsons]¹⁹.

**Multivariate Adaptive Regression Splines (MARS)** that combine basis functions placed over the input space, and are trained via a two-phase procedure that adds basis functions with the largest reduction of the training error, and then deletes the least important basis function at each time. In the evaluation, we use the implementation provided by [Jekabsons]²⁰.

**Queuing Models (QM)**¹¹ that simulate the behavior of an ideal multi-server single queue system that we implemented and manually configured to match the SGE setup.

These models implement different interpolation and regression techniques, and result in different response surfaces as exemplified in Figure 2. In particular, MLR models result in multidimensional planes, MARS models interpolate the data smoothly, and M5RT models partition the whole space into linear subspaces. QM models result in several planes with “jagged” boundaries that smooth away from one side of the model to the other, Kriging models interpolate the data smoothly, and Kriging+ models show

---

11. We use the model to predict the average response time of a single batch of \( N \) requests arriving within a 5 seconds interval while the system queue contains \( M \) jobs. In other terms, we are not predicting the average response time of the system subject to a steady state of requests with a given arrival rate. For this reason the QM graph in Figure 2 is linear and not exponential.
In this section we show that Kriging and Kriging+ models smooth sections when they use the queuing model and linear sections when they use the Kriging model.

5.3 Model Accuracy

We quantified the accuracy of the models referring to classic metrics, the mean absolute error (MAE), the root mean square error (RMSE) and the mean absolute percentage error (MAPE). These are defined by equations (6), (7) and (8), where $n$ denotes the number of validation samples, $y_i$ denotes the monitored output of the system, and $\hat{y}_i$ denotes the corresponding predicted value. Smaller values of the errors identify better models.

$$MAE = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i|$$ (6) 
$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2}$$ (7) 
$$MAPE = \frac{100}{n} \sum_{i=1}^{n} \frac{|y_i - \hat{y}_i|}{y_i}$$ (8)

In our experiments, we configured the models to predict the average response time of the SGE application (the output variable) given the number of running executor nodes, the amount of incoming jobs and the number of jobs queued in the system (the input variables).

We evaluated the predictions referring to the four usage profiles described at the end of Section 5.1. We did not use the data monitored while applying Profile 4 for training to obtain a completely unknown situation to deal with.

Table 1 summarizes the results. Column Model identifies the considered model, column Error identifies the error metric, columns Prof. 1-4 report the values of the error metrics for the four usage profiles. To simplify the reading we round the values to the second, and highlight the best (smallest) values of the error metrics for each profile in bold.

Unsurprisingly, splines perform the best in steady state conditions (Prof. 1), and their performances are closely accuracy of the models. During the runs we did not let the models update.

The Figure shows the response surfaces of the models considered in the evaluation process. The original models are four-dimensional, hence we projected them along one dimension (the system configuration) for visualization. The four dimensions are the amount of virtual machines, length of the queue, amount of incoming requests in the control period and average response time.

Fig. 2. The models for self-adaptive control considered in the evaluation process.
followed by Kriging+, Kriging and the other models. This result shows the excellent ability of MARS to capture trends in the data. However, accurately predicting the steady state behavior of cloud-based system has only limited relevance, since steady state conditions are seldom representative of the real working conditions that are usually characterized by both periodic fluctuations and sudden spikes [21].

The results with variable profiles indicate that Kriging+ and, to a lesser extent, Kriging perform the best. This confirms that Kriging and Kriging+ models can capture accurately the behavior of cloud-based systems under realistic working conditions, and are thus suitable to build effective cloud controllers.

An accurate analysis of the table leads to some additional considerations: (1) No model can completely eliminate prediction errors. This depends on the samples used for training and validation that were obtained from running systems, and are therefore inherently noisy and thus represent the real control conditions; (2) Kriging, Kriging+, queuing models and regression trees show better accuracy than splines under variable working conditions. Therefore, these models are more suitable to be used on-line, while regression splines should be the preferred choice for off-line modeling; (3) Queuing models perform well under many conditions, but their performance depends on the ability of modeling accurately the system at design time that may be difficult when dealing with complex cloud systems that can evolve over time; (4) Kriging+ models outperform all the other models under Profile 4. This confirms our hypothesis that Kriging+ models present the best results under realistic workload conditions. In the next section, we see that when Kriging models update on-line the difference between Kriging and Kriging+ models vanishes.

5.4 Models for Self-Adaptive Control

In this section we show the suitability of Kriging and Kriging+ models for controlling applications in the cloud by proving that Kriging and Kriging+ models (i) provide timely predictions that do not hinder the reactivity of the controller (RQ2 - promptness), and (ii) adapt to emerging application behaviors (RQ3 - adaptability).

Promptness of the Models

Model-based controllers may inspect models several hundred times during a single control cycle to compare possible control actions and their effects, thus the timeliness of the prediction becomes essential.

To evaluate the timeliness of the models, we compute the average time for both training and querying the models by means of K-fold cross-validation [8]. We ran the cross-validation for various values of K to study the impact of the number of samples (denoted by n) on the training time, querying time and accuracy expressed in terms of the cross validation error (CVE) computed as in (9). Small values of CVE characterize accurate models.

\[
CVE = \frac{1}{N} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{9}
\]

We executed the experiments using the datasets that we introduced in Section 5.3, and summarize the results in Table 2 for the steady profile and Table 3 for the variable profiles. To improve confidence of our results we repeated the experiments fifty times.

In both tables, the first column identifies the dataset in terms of the amount of sample data used for training the models (N), while the other columns report the average training time with the confidence interval (T), and the cross validation error (CVE) for each of the models. Values in bold highlight the best behaviors.

In this evaluation, we excluded both queuing models, because they are defined at design time and thus do not have training time, and multidimensional linear regression models, because of their unsatisfying results in the first experiment. Furthermore, we omit from the tables the query time since all the models behave comparably well, with average values for query time in the range of few milliseconds.

The tables highlight important differences in the promptness of the models while varying the amount and quality of the training samples. The training time indicates that all the models slow down while increasing the samples used for training, but with different values: Kriging, Kriging+ and M5 regression trees remain in the order of seconds, while MARS require over ten seconds. This may affect the controller depending on the sample cycles. In our experiments we used a typical sample cycle of 5 seconds that would negatively affect controllers based on MARS. The CVE presents relatively small differences between the models, with the exception of MARS models that present much better results for the steady profile. A low CVE is desirable, but does not balance the high training time of MARS models. The training time of MARS model depends on the stochastic nature of the model more than on the number of the samples, as reflected in the tables where the training time for the variable profile is lower than for steady profile.

These results confirm that Kriging, Kriging+ and M5 regression trees can be retrained online with negligible impact.
Adaptability of the Models

Self-adaptive controllers must adapt both to new data that can refine an initial non-optimal training set and to runtime changes in the controlled system. In this section we evaluate both how the models improve over time in absence of emerging behaviors, and how the models adapt in presence of sudden environmental changes.

We evaluate the model improvement over time by providing the controller with the stream of monitored data collected under Profile 4, and by periodically re-training the models. After each re-training we predict the behavior of the system and compare it against two validation sets built with the actual monitored data. We measure the quality of the predictions as the mean absolute error and the root square mean error. The first validation set corresponds to the whole trace, and we refer to it as Whole validation set, while the second validation set corresponds to the tail of the trace that follows the training data, and we refer to it as Future validation set. Comparing the prediction with the Future validation set measures the accuracy of the prediction, while comparing the prediction with the Whole validation set indicates the impact of the size of the training set.

We repeated the same process under different experimental settings by changing the frequency of model retraining, the amount of data that are considered for each input configuration and the aggregation strategy to preprocess the input data to update the training set. We use the strategy of no-model-adaptation as baseline for the comparison, and we considered the strategy of model-adaptation (i) using raw monitoring samples where we keep a limited amount of samples for each input location by replacing old samples with new ones, and (ii) using values over monitored samples averaged over a sliding window where we preprocess the monitored data by computing the average configuration that the system reached after executing the control actions.

To evaluate the adaptability to changing training sets, we started with the same initial training set, and accumulated the same input data over the experiment for all the models. We considered a long-lasting workload that includes non-periodic variations outside the training space generated by sequencing Profiles 2, 3 and 4.

Figure 3 shows the evolution of the mean absolute error for the different models over the Whole validation set. We can see that the MAE (Mean Absolute Error) stabilizes to a good value for all the models. This is due to a sufficiently rich and reasonably sized training set that we collected during the experiment. In fact, in this experiment we included in the training set also those observations generated according to Profile 4 that were not considered in the previous evaluations, and this explains the difference with the results reported in Table 1.

For small training sets, Kriging+ models behave better than the other models. This may represent an advantage in presence of frequent and sudden workload changes that may require recomputing the model with relatively small amount of new data.

Figure 5 also shows that the accuracy of some models did not improve monotonically. In particular, MRT5 and MARS present an oscillatory behavior. This depends on the training of both models that involves two-phase procedures based on non-trivial parameter optimizations. The optimization heavily depends on the current content of the training set, and might result in erratic behaviors for some distribution of training data.

To evaluate the adaptability of the models in presence of emerging behaviors, we simulated the emergence of a new system behavior characterized by a sudden increase of service time. We repeated the former experiment in presence of the sudden change with a periodic workload obtained by repeating Profile 2 six times.

Figure 4 shows the evolution of the mean absolute error over the Future validation set. The graph indicates that all the models present similar behavior. The value of the error slightly increases as the run proceeds up to the sudden change of the system behavior. At that time, there is a spike in the value of the error followed by a quick improvement, suggesting that the models adapt well to the new configuration.

5.5 Cloud Controllers

The experiments discussed in the previous sections show that Kriging and Kriging+ models fulfill the requirements about accuracy, adaptation and timing for on-line usage. In this section, we discuss how self-adaptive model-based

1939-1374 (c) 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
controllers based on Kriging and Kriging+ models perform under realistic working conditions.

We evaluated the performance of the controllers by deploying the SGE application on the cloud, and running our prototype implementation with different control policies. We generated synthetic workloads representative of different execution conditions commonly found in clouds according to Profiles 2, 3 and 4. We monitored both the performance of the application, to check if the average response time of the application violates the SLA, and the cloud, to measure the amount of resources allocated to the application. We measure the controller performance with three metrics [22]:

1) \( \overline{VM} \) that measures the average consumption of resources used during an experiment, and accounts for the cost of running the system under a given controller. Small values of \( \overline{VM} \) indicate efficient (cost-saving) controllers.

2) \( P \) that measures the number of SLA violations. The value of \( P \) is computed according to equation (10), where \( I_{SLA} \) is the Borel function defined by the SLA\(^{12}\), and \( n \) is the number of monitored instants. \( P \) measures the efficacy of controllers in an absolute sense, and its value should be equal to 0.

\[
P = \frac{1}{n} \left( n - \sum_{i=1}^{n} (I_{SLA}(i)) \right)
\]  

(10)

3) \( E \) that measures the average intensity of the violations defined as the distance of the average response time of the application when violations occur from the limits in the SLA. \( E \) is computed according to equation (11), where \( m \) is the number of monitored instants when an SLA violation occurs. \( E \) measures the \((n)\)effectiveness of a controller, and the best effectiveness corresponds to \( E = 0 \).

\[
E = \begin{cases} 
0 & \text{if SLA not violated}, \\
\frac{1}{m} \left| \sum_{i=1}^{m} I_{-SLA}(i) |y_i - y_{SLA}| \right| & \text{otherwise}
\end{cases}
\]

(11)

The results report also a summary value Total Penalty (TP) that combines these three metrics into a single value that merges the impact of the different metrics defined by equation (12).

\[
TP = \alpha \times \overline{VM} + \beta \times 100 \times P + \gamma \times E
\]

(12)

The parameters \( \alpha, \beta \) and \( \gamma \) weight the impact of the three metrics. In our experiments, we set decreasing penalties to resource consumption \( (\alpha = 5) \), SLA violations \( (\beta = 2) \), and the intensity of the violations \( (\gamma = 1) \). This choice of parameters gives priority to the cost of resource consumption over SLA violation, as common in public clouds where the cost of running several virtual machines over long periods is a concern.

We evaluate the generality of the controllers by repeating the same experiment under different settings of the cloud actuators and synthetic workloads. We create three experimental settings that vary from an ideal situation with slow varying workload and fast actuator (Experiment 1, VM instantiation in the order of seconds) to more realistic and challenging situations: in Experiment 2 we generate a fast varying workload and used slow actuator (VM instantiation in the order of minutes), which is compatible with private cloud settings, while in Experiment 3 we generate a fast varying workload and used slow actuator (VM instantiation in the order of minutes), which is compatible with popular public cloud settings. In this way, we study the performance degradation while moving from ideal to practical settings.

We evaluate the results of controllers based on Kriging and Kriging+ models by comparing them with the state-of-art controllers discussed early in this section: (i) Rule-based controllers that we configured according to the guidelines provided by our industrial partners (Sun at that time) in the context of the Reservoir EU Project [23], (ii) model-based controllers that use the same control logic as Kriging based controllers but different models, in particular the queuing and the MSRT model described earlier in this section.

During the evaluation we configure the controllers to adopt both reactive and proactive control strategies, when possible. We also design the controller to enable the use of different model updates and aggregation strategies that we used in the experiments. Not surprisingly, we obtain the best results with proactive control and with enabled update and aggregation strategies, and we report the results obtained under these optimal configurations only.

Table 4 summarizes the results of the experiments by reporting the metrics that we use to evaluate the controllers, \( \overline{VM}, P, E \), and the summary value \( TP \).

As expected, the first experiment (Experiment 1) does not reveal remarkable variations among the different models: Under stable workload conditions and with fast actuators, the control is quite simple and all approaches perform well. The second experiment (Experiment 2) indicates that most controllers behave well also in presence of fast varying workload conditions when they can rely on fast actuators. Only the controller based on the M5 regression trees presents some SLA violations (3%) with a non-negligible average violation time (3.37 seconds in average).

<table>
<thead>
<tr>
<th>Controller</th>
<th>Experiment 1</th>
<th></th>
<th></th>
<th></th>
<th>Experiment 2</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>Experiment 3</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( \overline{VM} )</td>
<td>( P )</td>
<td>( E )</td>
<td>( TP )</td>
<td>( \overline{VM} )</td>
<td>( P )</td>
<td>( E )</td>
<td>( TP )</td>
<td>( \overline{VM} )</td>
<td>( P )</td>
<td>( E )</td>
<td>( TP )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kriging</td>
<td>5.05</td>
<td>0.00</td>
<td>0.00</td>
<td>25.25</td>
<td>7.68</td>
<td>0.00</td>
<td>0.00</td>
<td>38.40</td>
<td>7.99</td>
<td>0.08</td>
<td>5.83</td>
<td>61.78</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kriging+</td>
<td>3.38</td>
<td>0.00</td>
<td>0.00</td>
<td>16.90</td>
<td>7.93</td>
<td>0.00</td>
<td>0.00</td>
<td>39.65</td>
<td>7.96</td>
<td>0.02</td>
<td>0.98</td>
<td>44.78</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSRT</td>
<td>3.93</td>
<td>0.00</td>
<td>0.00</td>
<td>19.65</td>
<td>7.63</td>
<td>0.03</td>
<td>3.37</td>
<td>47.52</td>
<td>8.22</td>
<td>0.04</td>
<td>3.16</td>
<td>52.26</td>
<td></td>
<td></td>
</tr>
<tr>
<td>QM</td>
<td>3.60</td>
<td>0.00</td>
<td>0.00</td>
<td>18.00</td>
<td>7.67</td>
<td>0.00</td>
<td>0.00</td>
<td>38.35</td>
<td>7.71</td>
<td>0.15</td>
<td>19.13</td>
<td>87.68</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rules</td>
<td>4.36</td>
<td>0.00</td>
<td>0.00</td>
<td>21.80</td>
<td>7.24</td>
<td>0.00</td>
<td>0.00</td>
<td>36.20</td>
<td>8.17</td>
<td>0.09</td>
<td>13.09</td>
<td>71.94</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\(^{12}\)The Borel function is 1 if the value of the output \( y_i \) is within the thresholds defined by the SLA \( y_{SLA} \), and is 0 if the SLA is violated.
The interesting results come from the third experiment (Experiment 3) that corresponds to the realistic case of fast varying workload in presence of slow actuators, and indicate clear differences among the controllers. While the average allocation of resources (VM) does not reveal major differences, the amount of SLA violations (P) and the average intensity of the violations (E) differ significantly, and indicate a clear advantage of Kriging+ over the other models. The average amount of SLA violation varies from 2% of Kriging+ to 15% of the queuing model (P). Only Kriging+ and M5 regression trees controllers present an amount of SLA violation below 5%, but the average intensity of an SLA violation using Kriging+ models is less than one second, while the average intensity of an SLA violation using M5 regression trees is over 3 seconds (E).

The advantage of controllers based on Kriging+ models over other models is well summarized by the total penalty (TP) value that is minimum for Kriging+. The parameters that we use for computing the TP value privilege the average resource allocation over the SLA violations. Different values of the parameters would lead to even more remarkable advantages of Kriging+ over the other models.

The relatively poor performance of controllers using pure Kriging models depends on the small training set used in the experiments that results in bad performances of the controllers for values far from the sampled values. The performances of Kriging based controllers increase with the availability of a larger set of training values and quickly approximates the performances of Kriging+ based controllers.

### 5.6 Summary of the results

The experiments discussed in this section highlight differences between the various models. Table 5 summarizes the results by indicating the suitability of the different models as shown in the various experiments. A (+) indicates that the model behavior matches completely the expectations, a (−) indicates that the model behavior is acceptable, a (− −) indicates serious deficiencies of the model and a (− − −) indicates that the experiment was not executed for the model because not applicable. We observe that rule-based (Rules), queuing (QM) and multi adaptive regression splines (MARS) models do not meet most of the requirements, while Kriging, Kriging+ and M5 regression trees (M5RT) models behave well. Among these last three models, Kriging+ and, to a lesser extend, Kriging outperform M5 regression trees. We thus conclude that Kriging models perform particularly well when used as the core component of self-adaptive controllers for the cloud.

### 6 Related Work

The problem of building efficient automatic cloud controllers has attracted the attention of several research communities, from control theory to software engineering and systems research. The many approaches proposed so far differ both in the models used to capture the knowledge of the system and in the strategies proposed to identify a suitable configuration while reacting to changes in the workload conditions [24]. We identify three main categories of automatic controllers: threshold- and rule-based, control-theoretic and autonomic (model-based) controllers. We provide an extensive discussion of autonomic control for the cloud in a recent survey paper [25], here we summarize the main characteristics of the different approaches focusing on techniques based on surrogate models, the closest to the work presented in this paper.

Threshold-based controllers are the current state-of-the-practice for industrial applications on public clouds thanks to their understandability and adaptability [26]. Rule-based controllers extend threshold controllers by introducing rules that trigger control events following the common ECA (event, condition, action) paradigm [23].

The Autoscale system by Gandhi et al. [27] strives to overcome the typical limitations of rule-based systems, including some of the linearity assumptions. Single-server experiments are used to derive the non-linear relationship between server load (defined as request rate multiplied by average request size) and number of requests in the system at steady state, as well as the maximum load sustainable by a single server without violating the SLA. However, some of the issues with rule-based systems remain, for instance with the assumption of pure linearity between system capacity and system resources and the lack of learning from emergent behavior.

Rules are commonly set manually assuming that (i) they can capture both the main phenomena and the characteristics of the system, (ii) the relations between the performance metrics and the amount of assigned resources are linear, and (iii) the system evolves only as predicted. These assumptions impact on the scope and reliability of rule-based approaches, since many systems do not satisfy these assumptions, especially in the cloud [28].

With respect to the approach proposed in this work, threshold and rule-based solutions sacrifice control efficiency for simplicity.

Control-theoretic approaches apply the principles of classic control theory to the problem of managing cloud applications. Most of the approaches proposed so far rely on dynamic models of the controlled system, and use standard control techniques, such as proportional, integral and derivative control, to synthesize the controllers [29]. The models are typically linear empirical models obtained by means of system identification.

The linearity of the models is required to formally demonstrate key properties of the controlled system, for instance its stability. Unfortunately computer systems rarely

---

**TABLE 5**

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Kriging</th>
<th>Kriging+</th>
<th>MSRT</th>
<th>MARS</th>
<th>QM</th>
<th>Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td></td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>++</td>
<td>++</td>
<td>+</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>−</td>
<td>−</td>
<td>− −</td>
<td>−</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>++</td>
<td>++</td>
<td>+</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Promptness</td>
<td>+</td>
<td>+</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td></td>
</tr>
<tr>
<td></td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>−</td>
<td>−</td>
<td></td>
</tr>
<tr>
<td>Adaptation</td>
<td>−</td>
<td>+</td>
<td>−</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>Controller</td>
<td>−</td>
<td>+</td>
<td>+</td>
<td>−</td>
<td>−</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>−</td>
<td></td>
<td>+</td>
</tr>
</tbody>
</table>

present linear behavior in practice. Control-theoretic controllers mitigate the imprecision induced by linear models by (i) adjusting linear coefficients over time [30], (ii) estimating them while the system is running [31] or (iii) deriving different models of the system and choose the one that fits best the observed behavior [32].

With respect to control-theoretic approaches our solution does not super-impose linearity in the behavior of the controlled applications, and does not assume monotonic relations between the control variables and the system outputs across the whole input space. It is sufficient to define the metrics that are relevant for the SLOs, and the controller will use performance metrics collected at runtime to automatically construct the system models to be used for the control objective.

Rule-based and control-theoretical approaches work well in presence of predictable changes in the workload conditions and stable systems that can be understood and modeled properly, but do not cope well with emerging system behavior.

**Autonomic controllers** solve the limitations of rule-based and control-theoretical controllers by adapting their behavior at runtime to match new execution conditions and variations in the system configuration.

Autonomic controllers differ in terms of techniques used to monitor, analyze, plan, act, and learn, and especially in terms of the knowledge representations [33]. We classify the different approaches as white- and black-box approaches according to the type of knowledge representation. White-box approaches use analytic models of the controlled systems and require knowledge of the system internals, while black-box approaches rely on information about the input/output behavior of the controlled system with little or no knowledge of the system internal details.

White-box controllers typically adopt queuing networks or layered queuing networks as their main analytic model. Such controllers differ for the specific queuing models used to represent the system behavior, the abstraction level captured with the models and the combination with other modeling features. Bi et al. [34] combine a M/M/c queuing model with several M/M/1 queuing models to control multi-tier applications. Dejun et al. [35] use a M/M/n/PS queuing model whose parameters are estimated at runtime. Woodside et al. [36] adopt a similar approach that is centered around a layered queuing network model.

White-box approaches work well when they can capture enough details of the system internals, and this becomes difficult in the context of the cloud that is opaque by design, and in presence of complex and evolving system configurations that may require a considerable effort to be modeled.

Black-box controllers derive models from the system behavior without requiring any knowledge of the system internals, thus overcoming the main limitation of white-box models.

An important class of black-box controllers rely on **surrogate models** that are built following a data-driven and bottom-up approach by approximating an unknown (n-dimensional) function via regression and/or interpolation on function samples. In their extensive analysis of surrogate modeling techniques for engineering purposes, Wang and Shan characterize surrogate models as parametric, non-parametric or semi-parametric [37].

Parametric models rely on a model structure that induces a finite set of parameters to be estimated. The choice of the model structure constrains the model flexibility, that is, its ability to approximate any functional shape. Parametric models studied in the context of autonomic controllers include artificial neural networks, response surface models and classic polynomial regression models. Nguyen et al. [38] use polynomial curve fitting to provide a black-box performance model of an application’s SLO violation rate for a given resource pressure. Maggio et al. [22] use artificial neural networks to build a model of system dynamics that can predict the system reaction to different resource allocations. Quiroz et al. use a Quadratic Response Surface Model to optimize the provisioning of virtual resources for jobs in Grid systems [39].

Non-parametric models do not constrain the number of configuration parameters used to fit the model to the modeled function. Trushkowsky et al. use statistical machine learning on data obtained through off-line controlled experiments to build performance models of storage systems [40]. Sharma et al. use logistic regression to predict SLO violations of applications running in private clouds [41].

Semi-parametric models combine different approaches to reduce the need of defining the values of many parameters, without relying on training only. Lama et al. [42] combine artificial neural networks and fuzzy logic to build self-adaptive controllers. The artificial neural network defines a suitable set of fuzzy rules, and the self-adaptive controller adapts the structure of the neural network at runtime, therefore automatically updating the fuzzy rules. Jung et al.’s controllers rely on (i) an estimator that uses automatic off-line experimentation to build a cost table that quantifies the performance degradation for each type of control action and workload, (ii) a layered queue network to predict the performance of each system configuration given a workload and (iii) an ARMA filter to estimate the duration of the stability of the current workload [43].

Controllers based on Reinforcement Learning (RL) learn directly optimal control policies via a trial-and-error learning strategy, and do not require a model of the system. Reinforcement learning solutions suffer from poor scalability in the action-state space and from long convergence rates. To alleviate these limitations, Tesauro et al. [44] combine RL with artificial neural networks and queuing networks, while Li and Venugopal [45] propose a distributed implementation of reinforcement learning to design self-adaptive controllers for the cloud.

### 7 Conclusions

In this paper we target the Infrastructure as a Service paradigm that offers computing resources to application providers as an on-line service, on-demand and through a pay-per-use model, and we focus on resource management. Managing resources at the infrastructure level must cope with two conflicting goals. On one side, allocating a large amount of resources to the applications reduces the risks of service violations. On the other side, reducing the amount
of allocated resources decreases the execution costs and optimizes resource consumption.

The sudden and unpredictable variations of the workload conditions that characterize many cloud applications reduce the efficacy of classic approaches that define the resource management policies at design time, and call for resource managers that adapt the resource management policies at runtime. The nature of the cloud that offers services, which abstract from the configuration details, limits the efficacy of cloud controllers based on white-box models of the controlled systems. Recent results indicate that controllers based on black-box, and in particular controllers based on surrogate models, can better meet the cloud requirements. The effectiveness of black-box controllers depends on the efficacy of the models in predicting the effects of resource allocations and workload variations on the quality of service.

In this paper we compared several controllers across different experiments and we showed the limitation of each design decision in increasingly realistic (and challenging) experimental settings. The experiments show that Kriging based controllers and their Kriging+ extension outperform other approaches when considering the combination of the interesting properties.

References


of allocated resources decreases the execution costs and optimizes resource consumption.

The sudden and unpredictable variations of the workload conditions that characterize many cloud applications reduce the efficacy of classic approaches that define the resource management policies at design time, and call for resource managers that adapt the resource management policies at runtime. The nature of the cloud that offers services, which abstract from the configuration details, limits the efficacy of cloud controllers based on white-box models of the controlled systems. Recent results indicate that controllers based on black-box, and in particular controllers based on surrogate models, can better meet the cloud requirements. The effectiveness of black-box controllers depends on the efficacy of the models in predicting the effects of resource allocations and workload variations on the quality of service.

In this paper we compared several controllers across different experiments and we showed the limitation of each design decision in increasingly realistic (and challenging) experimental settings. The experiments show that Kriging based controllers and their Kriging+ extension outperform other approaches when considering the combination of the interesting properties.

References


**Alessio Gambi** is a postdoctoral researcher at the Vienna University of Technology, Austria. He received the Laurea Degree in Computer Science from Politecnico di Milano (Italy) in 2006, the MSc Degree in Computer Science from the University of Illinois at Chicago (USA) in 2007 and the Doctorate Degree in informatics from the University of Lugano (Switzerland) in 2012. Research interests of Dr. Gambi include software engineering and autonomic computing applied to elastic cloud-based systems.

**Mauro Pezzè** is a Professor of software engineering at the University of Lugano and at the University of Milano Bicocca. He received his Laurea Degree in Computer Science from Politecnico di Milano (Italy) in 1984 and the Doctorate Degree in Computer Science from Politecnico di Milano (Italy) in 1989. The general research interests of Professor Pezzè are in the areas of software testing and analysis, autonomic computing, self-healing software systems, and service-base applications.

Prior to join the University of Milano Bicocca and the University of Lugano as full professor, Mauro Pezzè has been teaching assistant and associate professor at Politecnico di Milano, and visiting researcher at the University of Edinburgh and the University of California Irvine. Dr. Pezzè is associate editor of *IEEE Transactions on Software Engineering* (TSE) and member of the Steering Committee of the *International Conference on Software Engineering* (ICSE) and the *ACM International Symposium on Software Testing and Analysis* (ISSSTA). Mauro Pezzè is co-author of the book *Software Testing and Analysis, Process, Principles and Techniques* (Wiley, 2007), and he is the author or co-author of over 120 refereed journal and conference papers.

**Giovanni Toffetti** is a research associate at the IBM Haifa research laboratories, Israel. He received his Engineering Degree and his Doctorate Degree in Computer Science from Politecnico di Milano (Italy) respectively in 2001 and 2007. His research interests include model-driven engineering, content-based routing, model-based testing, autonomic and cloud computing.