Kriging Controllers for Cloud Applications

Alessio Gambi[1], Giovanni Toffetti [1], Cesare Pautasso[1], Mauro Pezzè[1,2]

[1]University of Lugano [2]University of Milano Bicocca

Abstract

Infrastructure as a Service is a Cloud computing paradigm that enables service providers to execute applications on third-party infrastructures with the pay-as-you-go billing model. Service providers can balance operational costs and quality of service by monitoring the application behavior and changing the deployed configuration at runtime as operating conditions change. Current approaches for automatically scaling cloud applications exploit user defined rules that respond well to events that can be predicted at design time, but have no provision for learning, hence do not react adequately to unexpected execution conditions.

In this article we present a new approach for designing autonomic controllers that automatically adapt to unpredicted conditions by dynamically updating a model of the system behavior. The approach demonstrates Kriging models as a suitable means to design efficient autonomic controllers.

1 Introduction

The Infrastructure as a Service (IaaS) Cloud computing paradigm provides the technical means for dynamically allocating resources so that service providers can scale their elastic applications by adding or removing virtual machines (VMs), while paying for the allocated resources following the *pay-as-you-go* billing model.

The execution conditions of services may vary greatly and unpredictably: Events like flash crowds, slash-dot effects, viral advertising campaigns and periodic trends can change the load considerably threatening the Quality of Service (QoS) of the applications. Thanks to IaaS, optimal service management can dynamically scale up the system to reduce the risks of QoS violations caused by load peaks. It can also scale down the system configuration to avoid wasting resources.

Common commercial solutions for automatically scaling services are based on statically defined rules that trigger reconfiguration actions when predefined events – often scoped only on low-level metrics – occur. For example, a rule may require the allocation of a new VM when the CPU usage exceeds an upper bound threshold. Rule-based approaches are intuitive and easy to imple-

Digital Object Indentifier 10.1109/MIC.2012.142 1089-7801/$26.00 2012 IEEE

ment, but limited by their static nature: Statically defined rules do not react well to unexpected execution conditions. Other traditional performance modelling approaches, such as linear models and simple queuing networks, also suffer from their static nature, unsuitable for controlling complex Cloud-based applications [2].

In this article, we present an original solution to dynamically scale the resources allocated to services to meet their QoS requirements avoiding resource over-provisioning. The solution does not rely on rules, rather it is based on black-box surrogate models of the system that evolve and improve over time, refers to metrics meaningful at the business-level that can be combined to treat control as a general optimization problem, and reacts properly to both expected and unexpected execution conditions. Our autonomic controller uses continuous learning to predict and manage service performance under different workloads. The controller requires a description of the service components, the monitored system metrics and the target performance levels. During the training phase, the controller correlates the system configurations with the monitoring variables and the QoS, and builds a model for each of the dimensions of the system behavior. During the control phase, the controllers use the models to scale the service while continuously learning from the service responses.

Differently from other approaches that are based on black box models, and that we survey in the next section, our approach uses Kriging surrogate models. Kriging models fit well with the requirements of highly dynamic control problems typical of Cloud-based services, as shown by the experiments reported in this paper.

The paper motivates the use of surrogate models for building controllers for the Cloud, discusses why Kriging models are useful in this context, and presents some experimental data collected with a prototype implementation of the controller. The experimental data support the claim that Kriging based controllers outperform current best practices.

2 Autonomic Controllers for Cloud applications

Autonomic controllers manage the allocation of resources to minimize execution costs while meeting QoS requirements. They work in a closed loop with the controlled system: monitor the system variables, analyze the monitored data, plan an optimal control strategy, and execute the strategy by means of control actions [5]. Controllers build and update some system knowledge, and use it to predict the system behavior in different configurations and running conditions, and to determine an optimal control strategy.

Some controllers represent knowledge without referring to particular models. Among these approaches the most interesting controllers are the one based on rules and reinforcement learning (RL). In their work, Jung et al. [4] discover rules via an off-line mining process, while Lim et al. [7] update rule triggering conditions that depend on the actual system configuration. Tesauro et al. [11] prove how RL controllers can learn optimal control policies for complex systems,

Digital Object Indentifier 10.1109/MIC.2012.142 1089-7801/$26.00 2012 IEEE

however they suffer from long training time and suboptimal performance in online learning, resulting in high costs.

Many more approaches use models to describe the steady state of the system, and can be divided in white-box or analytical models, and black-box or surrogate models. White-box approaches exploit many analytical models including classic queuing networks [13], layered queuing networks [6] as well as other models [9]. Analytical models are useful when the designers have a good understanding of systems' internal functioning, and the controllers can observe the systems from within. Black-box approaches build surrogate models by analyzing input-output data collected from running systems, thus, they refer only to the observable system behavior, and do not require extensive knowledge of the system internals.

The most relevant black-box approaches in the context of Cloud IaaS have been proposed by Bodik et al. [2] who use advanced regression models and classification techniques to describe system behavior and identify potential violations of service level agreements, while Malkowski et al. [8] use a case-based reasoning approach that considers past data to find suitable configurations.

In the context of Cloud computing, often IaaS users can access only partial information, for example, one can measure the CPU usage inside a VM, but cannot access the actual use of the physical CPU (or core). In this situation, the analytic models of white-box controllers may become too approximate or inaccurate, and may result in cost-ineffective control strategies. In these cases, black-box controllers are more effective.

Another challenge comes from the highly dynamic nature of Cloud computing, where both workload demands and resource availability fluctuate over time. In this situation, controllers must adapt frequently by quickly learning the effects of these ever-changing working conditions on the system behavior. In this context, static approaches (for instance, rule-based controllers) are of limited applicability, while others approaches (for example, reinforcement learning based controllers) become expensive both to be updated and to be brought to a steady state. As a consequence, we deem black-box approaches based on surrogate models to be the most suitable for controlling elastic applications in dynamic Cloud environments.

When choosing the specific black-box model for the controller we should consider: the cost of model training, the accuracy of model estimates, the knowledge and effort required in the configuration of the model and the ability to evaluate the quality of its approximation. The cost of training depends on the amount of input/output samples required to build an accurate model, and the time needed to build the model itself. Parametric models are faster to train because they rely on designers to decide on their internal structure, while in non parametric models the structure is obtained by the training data. This cost impacts on the learning ability of the controller: High training costs may result in the impossibility of building sufficiently accurate models. The accuracy of the model estimates and the possibility of over-fitting determine the responsiveness and precision of the control. Models like artificial neural networks or traditional regression models may be inaccurate if their "internal structure" (i.e., layers and neurons, model order) does not match the complexity of the problem. More-

Digital Object Indentifier 10.1109/MIC.2012.142 1089-7801/$26.00 2012 IEEE

over, they offer no estimation of the quality of the function approximation they provide.

We propose to use Kriging models because – as we discuss in [12] – they can be trained quickly enough to be applied on-line, do not require designers to define their internal structure, provide a good accuracy also with small training datasets, and estimate the confidence of their predictions. This last information is valuable and allows controllers to drive their training phase and to balance the risks at control time.

3 Kriging models

Kriging models are spatial data interpolators akin to radial basis functions. Kriging models extend traditional regression with stochastic Gaussian processes [14]. As summarized by Jones et al. [3], standard linear regression and Kriging models share a common mathematical framework that includes regressors and errors, but put different emphasis on the two aspects: Linear regression focuses on the regressors and their coefficient estimates, and makes simplistic assumptions about the errors and their independence. Kriging modeling focuses on the correlation structure of the errors, and makes simplistic assumptions about the regressors. In other terms, linear regression is about estimating regression coefficients that (together with the assumed functional form) fully describe *what the function is about*, while Kriging modeling is about estimating correlation parameters that describe *how the function typically behaves*, thus describing how much the function tends to change while moving along different directions.

Kriging models predict the value of the target function in unknown coordinates by computing the value that is *most consistent* with the estimated typical behavior of the function, which is expressed as a weighted combination of the known values of the function itself, known as the training data.

Kriging modeling assumes that the errors are *correlated*, and that this correlation relates to the distance between the corresponding sampled points: The correlation is high when the corresponding points are close and low when they are far apart. In each dimension, the correlation function is governed by parameters that describe the sensitivity of the function, and that capture the seamlessness of the function behavior in response to changes in the input variable. Kriging models derive these parameters by maximizing the likelihood (MLE) of the sample within the average value of the function and the variance of the error.

The correlation of the errors affects the estimate of the accuracy of the predictions that depends on the distance between the coordinates of the prediction and the coordinates of the known samples. In Kriging models, the uncertainty of the prediction is commonly measured as the root mean squared error (RMSE), which is zero at sampled coordinates, approximates the standard deviation σ of the error at coordinates far from the sampled ones, and lies in the range between zero and σ at intermediate coordinates. Therefore, the prediction is the sampled value when available, the estimated mean of the function when far from

Digital Object Indentifier 10.1109/MIC.2012.142 1089-7801/$26.00 2012 IEEE

sampled coordinates, and a smooth interpolation of the data at intermediate coordinates.

Practical evidence shows that the models estimated by means of MLE approximate the original functions quite well even with a limited number of samples, and indicates that for continuous and smooth functions, few samples are sufficient to obtain a good approximation [1].

4 A Kriging-based controller

We designed a Kriging-based controller to enhance the capabilities of Cloud infrastructures and we provided a prototypical implementation written in Java. The implementation follows a *plug-in* architectural pattern so controller components can be easily replaced. At the moment, we have implemented plug-ins that use the Octave engine[1] for building the Kriging models, and that bind to the Amazon EC2, Eucalyptus, and Reservoir Cloud infrastructure APIs[2] for receiving monitoring data (monitor interface) and sending scaling actions (deployment interface).

The Kriging-based controller follows the reference architecture for autonomic controllers adapted for Cloud computing (Figure 1). The controller monitors the incoming workload W, for example the number of requests issued in the last control period, the performance indicators at the application level P, such as the average response time, and the current system configuration SC, for instance how many virtual machines execute the service. It may change the system configuration with control actions A, for example, it may instantiate or de-instantiate virtual machines.

The controller is composed of a *model builder*, one *Kriging model* for each target QoS concern (Service Level Objective), and a *decision maker* that in parallel implements two activities: the control loop and the model training.

In the control loop, the decision maker queries the Kriging models using workload and system configuration data to predict the performance evolution ($^*P$). Then, the decision maker compares the estimated performance against the target QoS, and decides the corrective actions to be performed. The controller derives the target configuration of the system ($^*SC$) expected to meet the target QoS in response to the monitored workload ($^*W$) by querying again the surrogate models. Once the decision maker has chosen a target configuration, it also derives the set of control actions for the Cloud.

When the control goal includes several objectives, finding a suitable configuration becomes a multi-objective optimization problem that can be solved in many ways, for instance, by ranking objectives, computing a single aggregated objective function (AOF) or following Pareto optimization methods.

Each model is trained online, in parallel with the control loop, aiming to maintain the Kriging models updated. The model builder receives raw monitoring data and elaborates them to produce the samples that are used to update the

[1] http://www.gnu.org/s/octave/
[2] http://www.reservoir-fp7.eu/

Digital Object Indentifier 10.1109/MIC.2012.142 1089-7801/$26.00 2012 IEEE

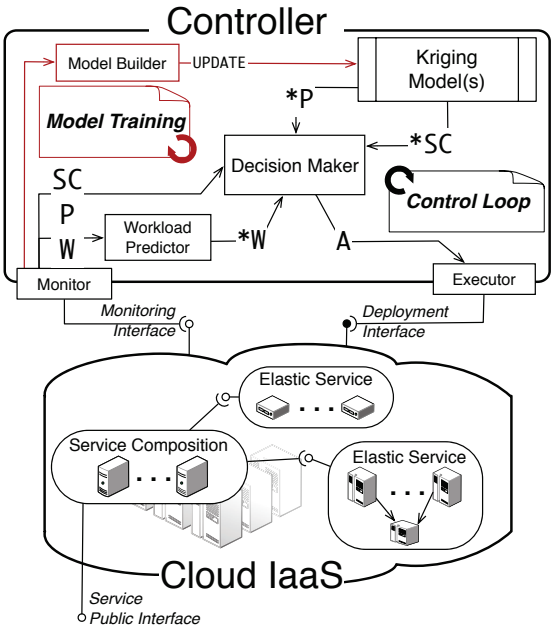


Figure 1: Kriging-based controller in a Cloud IaaS architecture

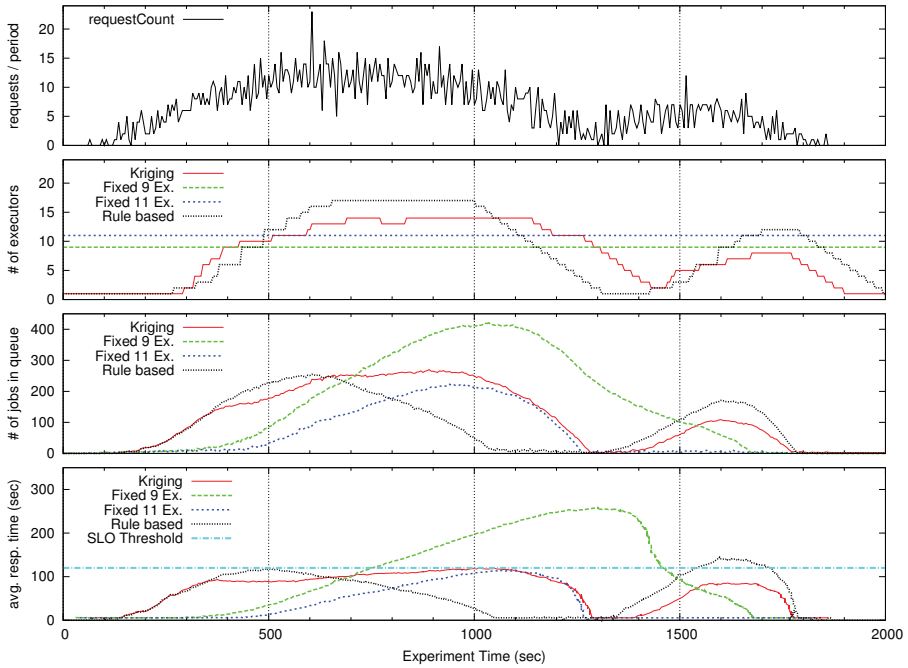
models. When needed, for example when the system reaches unvisited configurations, or when model estimation accuracy drops below a given threshold, the model builder triggers a retraining. In our initial implementation, we trained surrogate models with samples obtained from experiments with the system in steady-state, and we used the models at runtime to predict the system performance for instantaneous values of the input values like system configuration and workload.

5 Experimental Validation

In this section we provide experimental evidence of the efficacy of using Kriging controllers for scaling applications deployed on Cloud infrastructures. The results reported confirm the preliminary results obtained with small datasets and different applications [1, 12].

We validated a prototype implementation of a Kriging-based controller by applying it to manage an elastic application based on the Sun Grid Engine (SGE) middleware. SGE follows a standard Grid computing architecture with a singleton master node and a set of executor nodes: The master receives jobs that are dispatched to the executors that run them. The middleware has been deployed in a Cloud infrastructure, where virtual execution nodes can be allocated dynamically. In the case study, the Kriging model correlates the average system response time with the number of incoming and queued jobs, and the

Digital Object Indentifier 10.1109/MIC.2012.142 1089-7801/$26.00 2012 IEEE



The results were obtained running the Reservoir architecture on a 6 quad-core CPU (Intel Xeon @ 2.00GHz) machine equipped with 128 GB RAM, running Ubuntu 11.04 64-bit server. The VM running the SGE master node runs with 2 GB RAM, 2 virtual CPUs and 2 virtual network interfaces: one for receiving jobs and one to dispatch them to executors; the VMs running the SGE executors have 1 GB RAM, 1 virtual CPU and one network interface. In this setting, the largest configuration we are allowed consists of the master node and 22 executors. Similar results were obtained deploying on Amazon EC2.

Figure 2: Configuration, queue, and response time evolution for the SGE using Kriging-based, rule-based, and fixed control

system configuration (i.e., the number of executors nodes). The controller monitors these metrics, and adds or removes nodes to maintain the average system response time under a specified threshold.[3]

In Figure 2 we report the results of different runs of the system subject to an identical workload. The workload lasts thirty minutes and fluctuates in time; it consists of video conversion jobs that (on average) execute in six seconds. As specified by the SLO over the response time, the system is required to complete jobs in less than two minutes in average. The controller changes the number of executor VMs in order to respect the SLO.

The plot depicts the workload in terms of requests per period, number of active executors, number of jobs in the system, and measured average response time for four different runs. The continuous (red) line represents the system

[3] A video that shows a live run of the controller is available at `http://www.youtube.com/user/STARResearch`.

Digital Object Indentifier 10.1109/MIC.2012.142 1089-7801/$26.00 2012 IEEE

Control	Avg. VMs	% Viol.	Avg. Viol. (s)
Fixed 9	9	35%	85.8
Fixed 11	11	0%	0
Kriging	8.52	0%	0
Rules 5-15	8.43	9%	13.1

Table 1: Quantitative comparison of different control logics

under the control of our Kriging-based controller. We can see how the number of executors changes over time and how the average response time is always below the 2-minute threshold (light-blue line in the figure). For reference, the dashed (green) line represent a statically configured system with the same total VM/resource allocation cost[4] as the Kriging solution. In this case the SLOs are violated. The dotted (blue) line represents the smallest[5] fixed configuration able to withstand the given workload without violating the SLOs. Finally, the fine-dotted (black) line represents the behaviour of a state-of-the-art rule-based controller as presented by Rodero-Merino et al. in [10]: We set the scaling up rule threshold for the queue length at 15 jobs per executor, meaning that each time the number of jobs in queue divided by the number of active executors exceeds this threshold a new executor VM instance is spawned, and we set the scaling down threshold to 5 jobs per executor.

As expected, dynamically adapting to the workload yields a saving in resource usage while reducing SLO violations. Table 1 presents a quantitative summary of the results obtained with the different control logics. The table reports the average number of virtual machines used in the run, the percentage of time in which violations occurred, and the average magnitude of the violation in seconds. The reported results show that the proposed approach is able to provide automatically derived control logics that prevents SLO violations while managing the resources efficiently, while other approaches may require manual tuning, as discussed below.

With additional knowledge on the workload or its characteristics (for example, average, variance, variation, periodicity), and on the system and actuator dynamics, one could optimize the thresholds towards obtaining the best possible performance for the rule-based control and the given workload. This optimization is typically performed manually and off-line by an expert, but it would need to be repeated if the workload changed.

We chose SGE deliberately as a use case for comparing with rule based approaches, since the system response time can be tolerably approximated with a linear function of the jobs in queue, and thus rule-based approaches perform well. The particular values of the rule thresholds that we picked show the limits of assuming a linear behavior of the system at hand. For instance, the scaling up rule is triggered quickly enough to react to the first peak in the workload, but not enough to react to the second peak avoiding violations. If we were to lower

[4]We assume a pay per use model for each VM computed in seconds
[5]We found it using a trial and error approach.

Digital Object Indentifier 10.1109/MIC.2012.142 1089-7801/$26.00 2012 IEEE

the scaling up threshold, we would obtain a more reactive controller that would avoid violations for the second peak but would use more resources. Raising the scale down threshold would instead lower resource usage increasing the chance of control oscillations.

The bottom line is that the rule-based controller does not know that, for the considered SGE implementation, very large configurations have a high cost but yield a limited positive effect on the system response time, hence it is preferable to use a medium-sized configuration early on to reduce costs. The only way to know this is by observing the system behavior at runtime and storing this knowledge in an appropriate model. This model is what our proposed controller uses to choose the "most beneficial" configuration according to some optimization goal.

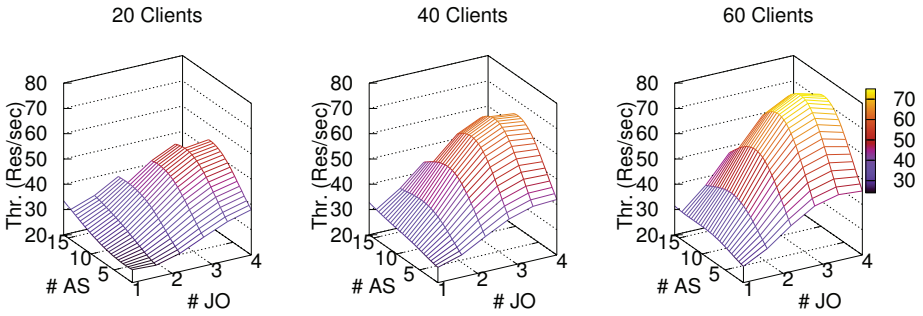


Figure 3: Kriging model of the throughput of the DoReMap system as a function of the number of clients and the configuration in terms of JOpera composition engine (JO) and Doodle application server (AS) VM instances

Many cloud-based applications exhibit a complex behavior that cannot be adequately approximated linearly and is not even guaranteed to be monotonic. Consider for instance Figure 3, depicting the Kriging model for the throughput of the DoReMap service composition we presented in [12]. Since adding virtual machines to the system is actually detrimental above a certain point, deriving a set of appropriate scaling rules for these applications is a very challenging task.

Our approach is capable of building a model of the deployed system and automatically implementing a control policy achieving goals expressed in terms of business metrics (e.g., minimize the application's operational cost while avoiding violations). With this respect it is therefore a *more general and expressive* solution than rule-based control. Moreover, by operating in closed loop, it is able to measure the effects of different control actions and working conditions, providing support for *self-adaptivity*.

Digital Object Indentifier 10.1109/MIC.2012.142 1089-7801/$26.00 2012 IEEE

6 Conclusions

In this paper we presented a new approach to design autonomic controllers based on Kriging models together with a prototypical implementation. The paper discusses the suitability of black-box controllers based on surrogate models, proposes Kriging as a suitable type of surrogate models, and indicates the main characteristics of a prototype implementation of a Kriging controller that was used in our experiments. The paper presents experimental data obtained with the prototype implementation on the Sun Grid Engine, an industrial case study used as benchmark in the community. Our data confirm that Kriging based controllers are able to implement control logics that are comparable to state-of-the-art rule-based control, but provide an *automatic*, *more general and expressive* approach to system performance modelling as well as *self-adaptation capabilities*.

References

[1] A.Gambi, M. Pezzè, and G. Toffetti. Protecting SLA with surrogate models. In *Proc. Principles of Engineering Service-Oriented Systems*, pages 71–77. ACM Press, 2010.

[2] P. Bodik, R. Griffith, C. Sutton, A. Fox, M. Jordan, and D. Patterson. Statistical machine learning makes automatic control practical for internet datacenters. In *Proceedings of the Conference on Hot topics in cloud computing*, HotCloud'09, pages 75–80, 2009.

[3] D. Jones, M. Schonlau, and W. Welch. Efficient global optimization of expensive black-box functions. *Global optimization*, 13(4):455–492, 1998.

[4] G. Jung, K. Joshi, M. Hiltunen, R. Schlichting, and C. Pu. Generating adaptation policies for multi-tier applications in consolidated server environments. In *Proocedings of the International Conference on Autonomic Computing and Communications (ICAC'08)*, pages 23–32, June 2008.

[5] J. Kephart and D. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[6] J. Z. Li, J. W. Chinneck, C. M. Woodside, and M. Litoiu. Fast scalable optimization to configure service systems having cost and quality of service constraints. In *International Conference on Autonomic Computing*, ICAC'09, pages 159–168, 2009.

[7] H. C. Lim, S. Babu, and J. S. Chase. Automated control for elastic storage. In *Proceeding of the International Conference on Autonomic Computing*, ICAC'10, pages 1–10, 2010.

Digital Object Indentifier 10.1109/MIC.2012.142 1089-7801/$26.00 2012 IEEE

[8] S. J. Malkowski, M. Hedwig, J. Li, C. Pu, and D. Neumann. Automated control for elastic n-tier workloads based on empirical modeling. In *Proceedings of the 8th ACM international conference on Autonomic computing*, ICAC'11, pages 131–140, June 2011.

[9] M. N. Bennani and D. A. Menasce. Resource allocation for autonomic data centers using analytic performance models. In *Proceedings of the International Conference on Automatic Computing*, ICAC'05, pages 229–240, 2005.

[10] L. Rodero-Merino, L. M. Vaquero, V. Gil, F. Galán, J. Fontán, R. S. Montero, and I. M. Llorente. From infrastructure delivery to service management in clouds. *Future Generation Computer Systems*, 26(8):1226–1240, Oct. 2010.

[11] G. Tesauro. Reinforcement learning in autonomic computing: A manifesto and case studies. *IEEE Internet Computing*, 11(1):22–30, 2007.

[12] G. Toffetti, A. Gambi, M. Pezzè, and C. Pautasso. Engineering autonomic controllers for virtualized web applications. In *Proc. Int'l Conf. Web Engineering*, pages 66–80. Springer-Verlag, 2010.

[13] B. Urgaonkar and A. Chandra. Dynamic provisioning of multi-tier internet applications. In *Proceedings of the International Conference on Automatic Computing*, ICAC'05, pages 217–228, 2005.

[14] W. van Beers and J. Kleijnen. Kriging interpolation in simulation: a survey. In *Proc. Winter Simulation Conference*, pages 113–121. Institute of Electrical and Electronics Engineers, University of Michigan, 2004.

Alessio Gambi is a PhD student at the University of Lugano, Switzerland. His research interests include Software Engineering applied to autonomic and cloud computing. Gambi has a M.Sc. in Computer System Engineering form the Politecnico di Milano, Italy. Contact him at alessio.gambi@usi.ch.

Dr. Giovanni Toffetti is a research associate at the University of Lugano, Switzerland. His research interests include model driven engineering, content-based routing, model-based testing and autonomic computing. Dr. Toffetti has a PhD in Computer Science from the Politecnico di Milano, Italy. Contact him at toffettg@usi.ch

Dr. Cesare Pautasso is an assistant professor at the University of Lugano, Switzerland. His research interests include software architecture and RESTful service composition. Dr. Pautasso has a PhD in Computer Science from ETH Zurich. Contact him at c.pautasso@ieee.org.

Digital Object Indentifier 10.1109/MIC.2012.142 1089-7801/$26.00 2012 IEEE

Dr. Mauro Pezzè is a professor of Computer Science at the University of Milano Bicocca, Italy, and at the University of Lugano, Switzerland, where he serves as dean of the Faculty of Informatics. His research interests include software testing and analysis, autonomic computing and self-healing software systems. Contact him at mauro.pezze@usi.ch.

Digital Object Indentifier 10.1109/MIC.2012.142 1089-7801/$26.00 2012 IEEE