Engineering Autonomic Controllers for Virtualized Web Applications

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

[1] University of Lugano
6904, Lugano, Switzerland
[2] University of Milano Bicocca
20126, Milan, Italy

Abstract. Modern Web applications are often hosted in a virtualized cloud computing infrastructure, and can dynamically scale in response to unpredictable changes in the workload to guarantee a given service level agreement. In this paper we propose to use Kriging surrogate models to approximate the performance profile of virtualized, multi-tier Web applications. The model is first built through a set of automated and controlled experiments at staging time, and can be later updated and refined by monitoring the Web application deployed in production. We claim that surrogate modeling makes a very good candidate for a model-driven approach to the engineering of an autonomic controller. Our experimental evaluation shows that the model predictions are faithful to the observed system's performance, they improve with an increasing amount of samples and they can be computed quickly. We also provide evidence that the model can be effectively used to synthetize an aggregated objective function, a critical component of the autonomic controller. The approach is evaluated in the context of a RESTful Web service composition case study deployed on the RESERVOIR cloud.

1 Introduction

More and more Web applications are hosted in Cloud computing environments to reduce their operational and maintenance costs. Cloud infrastructures build upon virtualization technology to simplify the deployment of Web applications and to enable application resources to be controlled dynamically [12]. Clouds offer the necessary flexibility to scale Web applications in order to support a variable number of clients during the runtime. These capabilities need to be balanced against increasing performance overhead and architecture complexity. Whereas the performance overhead may be acceptable for many applications [16], the problem of finding suitable deployment configurations of virtualized Web applications facing unpredictable client demand changes remains open.

In this paper we focus on multi-tier Web applications that are executed within virtualized infrastructures, and propose a method for automatically reconfiguring these applications in response to sudden and unpredictable changes in client workload that may derive for example from flash crowd or periodic peaks [2].

B. Benatallah et al. (Eds.): ICWE 2010, LNCS 6189, pp. 66–80, 2010.

Introducing resource virtualization technology in infrastructures that host Web applications requires both to determine how many replicas of the Web application components to instantiate, and to address many details that include: how to assign each virtualized component to the physical resources, how to size the resources (CPU and memory) allocated to each virtual machine, how to bind service replicas with one another, and how to optimally distribute the client workload over a heterogeneous set of resources. Thus, the layer of abstraction introduced by virtualization in the Web application architecture augments the set of possible configuration decisions, and makes it very difficult to predict the effects of reconfiguration actions on the overall system performance.

To address these problems, we propose a model-driven approach to engineer an effective autonomic controller. Our models capture the relationship between many tunable configuration parameters and the expected performance of the Web application. In particular, we show how to apply multi-dimensional surrogate models [19] to study and predict the performance of virtualized Web applications. As more samples are observed, the prediction error of the models is reduced. We use the model to construct a utility function that can drive the controller self-configuration decisions. As opposed to other modeling approaches, the advantages of using surrogate models in this context are manifold. First, surrogate models are independent from the actual system complexity. As such, they can quickly predict the expected system behaviour [19]. They provide confidence measures that indicate the possible directions to follow when searching for optimal configurations. They effectively deal with highly-dimensional configuration spaces, and can be quickly updated at runtime. Thus, they support a continuous learning and prediction improving process. We use surrogate models to bridge the gap between measured non-functional system properties, like responsiveness, availability, and throughput, and the system configuration, to approximate the complex and unknown relation between them.

The rest of this paper is structured as follows. Section 2 defines the context and the architecture of the autonomic controller. Section 3 describes the case study use in this paper. Section 4 validates the approach experimentally by discussing the main research questions, the experimental methodology, the experimental results and their effective adoption for control. Section 5 outlines the related work. Finally, Section 6 summarizes the main contribution of this paper and delineates future research directions.

2 Architecture

Figure 1 shows the key elements of a virtualized Web application, and sketches a high-level representation of the controller architecture. The controlled system is composed of the virtualized Web application that runs on top of the physical hardware and network infrastructure. The system is designed with a public service interface and an internal management interface. The public service interface is used by the clients of the Web application, while the management interface enables both the monitoring of the system configuration (SC) and its performance (P) and the application of reconfiguration control actions (A).

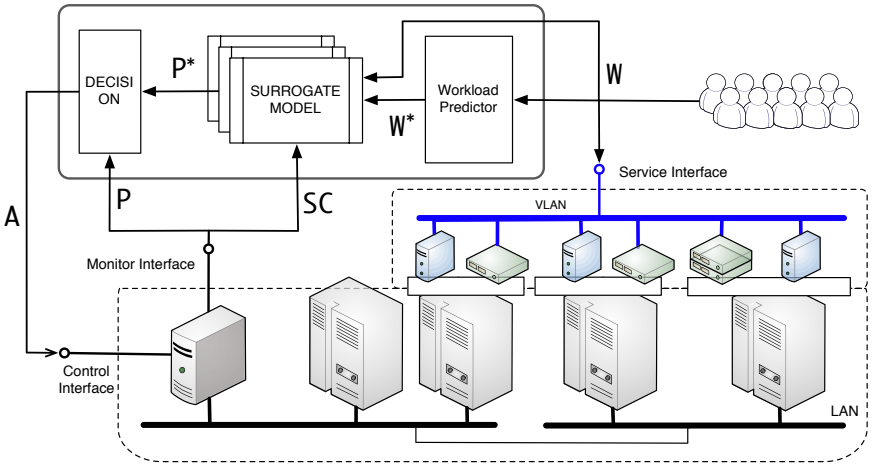


Fig. 1. Logical architecture of an autonomic, virtualized Web application

In production, the system is subject to a varying client workload (W) that can be characterized according to different dimensions (for instance, average inter-arrival times of request per request type, average request size, workload mix). We assume that the controller does not limit admission to the services and thus the workload cannot be altered by the controller. However, the controller could be fed both with information about the current workload (W) and the predicted future workload (W*). In control-theoretical terms, the workload is generally represented as a *disturbance*, that is a non-controllable system input [22]. The controller internal representation of the system is kept up-to-date by monitoring both layers of the controlled system. Monitoring data (the *system output*) coming from both the virtualized Web application components and the underlying physical infrastructure include the current system configuration, environmental information, as well as key performance indicators (KPIs) such as workload and performance measurements (for example, response times, throughputs and SLA violations). The KPIs are commonly used to express the goals (or service-level objectives, SLOs) of the controller. The controller aims to determine the optimal system configuration (*control input*) that meets the goals required to ensure that the performance of the Web application is acceptable.

Autonomic control approaches are characterized by the so-called MAPE-K closed-loop model, named after the basic activities that comprise the loop: Monitor, Analise, Plan and Execute with Knowledge [3,8]. Autonomic controllers are driven by control policies that express their main goals. For instance a control policy may give highest priority to preventing SLA violations, and, once SLAs are guaranteed, it may minimize the operational costs. Controllers *monitor* the systems key performance indicators that characterize the service quality. They *analyse* the collected data to diagnose for instance (potential) SLA violations. They *plan* a strategy to meet the control goals using *knowledge* about the

expected system behaviour, usually expressed as models. They *execute* the control actions that implement the strategy. Finally, they evaluate the effects of the control actions updating their knowledge.

In a Cloud computing scenario, the possible control actions are amenable to virtual machine instantiation, de-instantiation, and placement, as well as setting the system to a specific point of its *configuration space*. In this paper, we focus on the critical *Knowledge* component of a controller. This component provides the configuration analyzer with the essential information to self-configure the virtualized Web application, as it contains the representation of the system used to find the appropriate configuration for a specific goal.

We capture the essential information of the knowledge component with surrogate models, that are mathematical approximations of unknown complex functions built from sampling [19]. Surrogate models provide both a predicted value and an accuracy measure of the actual function. They are widely used in engineering, when the sampling process is expensive, the exploration of the complete design space is not feasible, and an upper-bounded approximation is tolerable. For example, surrogate models are often used to reduce the highly expensive computer simulations or controlled experiments when exploring a vast design or configuration space.

Among many possible surrogate models, we use Kriging models [18] that interpolate the space with Gaussian processes to approximate the system behavior sampled through controlled experiments. Kriging models fit well our problem domain for several reasons. Modern Kriging provides an *exact* sample interpolation, i.e., the predicted outputs for the inputs that correspond to the samples match the measured outputs. Having a model that matches the samples exactly is important for the reliability of the predictions, and is often required in computer aided engineering. Kriging models cover the whole parameter space (the experimental area), thus they often produce better predictions than regression analysis [18]. Also, thanks to their excellent performance properties, Kriging models can be efficiently used to build controllers for run-time adaptation of the Web application configurations. As we show in this paper, Kriging models can be computed quickly, also in presence of many samples, and thus can deal with frequent updates.

3 Case Study

In our investigations we consider a composite RESTful Web service called Doodle Restaurant Map (DoReMap) [13]. DoReMap manages voting polls that are mashed up with a well-known map widget and facilitate agreements on nearby restaurants.

Users query the service for restaurants in the vicinity of a particular location. The service identifies a set of restaurants, and uses it to automatically create a voting poll so that users can cast their vote for a particular restaurant. To ease the choice, the voting poll is augmented with a map that shows the location of each restaurant.

Fig. 2. The Doodle Restaurant Map (DoReMap) Web Application

Figure 2 visualizes the three main steps comprising the use of this service: (1) A user playing the role of *initiator* submits a poll creation form; (2) the system creates the poll and sends back to the initiator a *participation link*; (3) the initiator communicates the participation link to the other *participant* users that contact the poll, look at the restaurant locations on the map, and cast their votes. Once enough participants express their choices the poll is closed and no more votes can be submitted. The result of the poll might be inspected through the original participation link until the *initiator* deletes the poll from the system. We assume that DoReMap should comply with a simple SLA that specifies the maximum response time for each of the application requests, such as creation of the poll resource and vote.

3.1 Service Composition Model

The DoReMap service composes two atomic RESTful Web services: a restaurant search service, inspired by Yahoo! Local search engine API [1], used to query for restaurants near a given location, and a voting poll service inspired by the Doodle REST API[2], used to create, update and close polls.

The composition is modeled using the JOpera visual composition language [14], and is executed on the JOpera Engine version 2.4.9[3]. Figure 3 shows the control flow view of the JOpera composition model, limited to the creation of the poll and the handling of the client vote requests. The composition receives the input parameters from the initiator's form and then invokes the restaurant search service. Once the search is complete, the composition uses the data about the restaurants to create both the map that shows their location and the poll by invoking the voting poll service. When both steps have completed, the page hosting the poll mashup for the participants is generated and stored at a unique URI. This URI is then embedded as the participation link into the notification page that is sent back to the initiator.

[1] `http://local.yahoo.com`
[2] `http://www.doodle.com`
[3] `http://www.jopera.org`

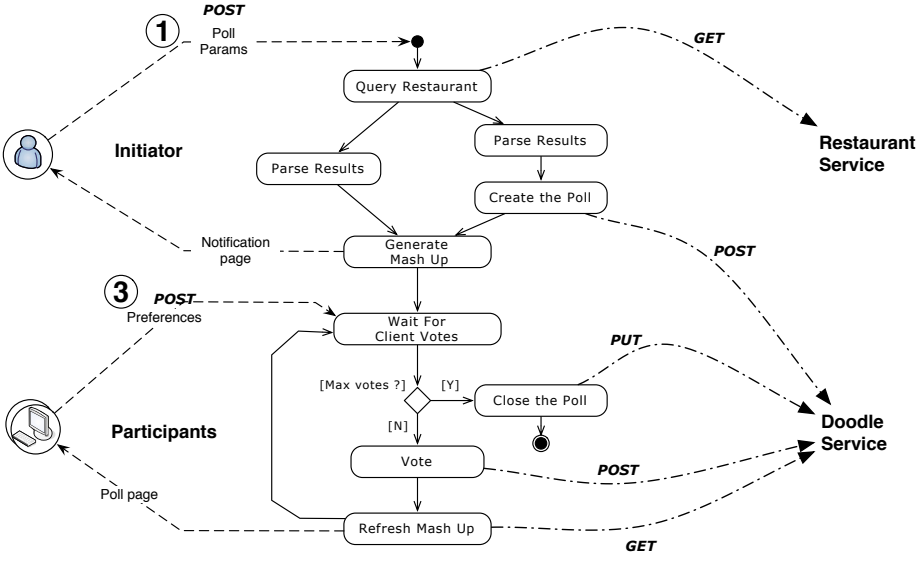


Fig. 3. Control flow model of the DoReMap service composition

Having completed the initialization stage, the composite service enters the main loop stage to collect the votes of the participants. At each vote, the composition checks the status of the poll service and updates it accordingly. When the status of the poll changes, the composition updates the poll page. The composite service continues executing until the number of votes reaches a threshold given by the initiator. At this point, the DoReMap service closes the poll, and keeps its final state published until explicitly deleted by a client.

3.2 System Architecture and Deployment

The architecture of DoReMap includes several components that are deployed inside four virtual servers interacting through a virtual network. As Figure 4 shows, the JOpera engine is deployed on its own virtual server (`JOperaAS`) to separate the logic implementing the composition from the component atomic services. Both sets of atomic services are designed as standard two-tier Web services and are composed of a REST front-end and a database back-end. They are deployed following different policies: the restaurant lookup service components are packaged into a single server, called `RestaurantAS`, because they are used only to read data, and because they should optimize the access to the data during the search; the voting services instead are deployed inside two separated servers, called respectively `DoodleAS` and `DoodleDB`, to separate the data access logic from the database tier.

Architecting the Web application as a loosely coupled composition of services deployed on separate virtual servers increases flexibility when it comes to deploying the composite service in the cloud. Each virtual server is packaged as a

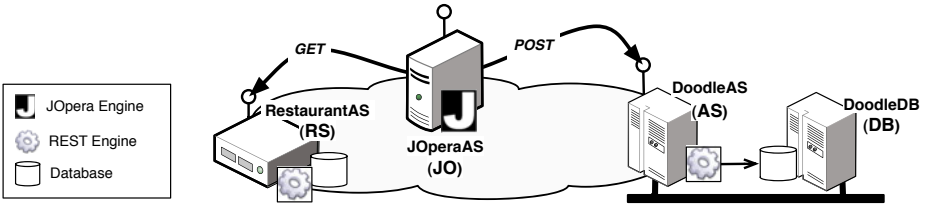


Fig. 4. Logical Architecture of the DoReMap Composite Service

disk image that can be seamlessly instantiated as virtual machine in the Cloud. To serve a demanding workload, multiple instances of critical services can be dynamically created without replicating the entire Web application [21]. For example, a growing number of concurrent clients might increase the number of requests at the composition layer. To prevent service saturation, new instances of JOpera components can be added to serve all requests without violating the SLA. Similarly, the system can respond to a changing workload mix by adding new `DoodleAS` and `RestaurantAS` server instances, thus scaling *horizontally*.

3.3 The RESERVOIR Cloud Computing Testbed

We executed the virtualized version of the DoReMap component services on an infrastructure developed within the FP7 RESERVOIR Project[4].

We executed the service on a partition of the RESERVOIR testbed cloud composed of six Blades IBM LS21 biprocessor dual-core Opteron 2218 at 2.6GHz with 8GB RAM DDR2 at 667MHz divided into two separated sites of three machines each, and linked by a dedicated high speed network. One of the machines was devoted to infrastructure services such as deployment and monitoring, while the remaining ones were used as raw resource pools, running the Web application virtual servers.

The RESERVOIR cloud infrastructure is designed to support run time deployment and live migration of virtual machines. This enables virtualized Web applications to be dynamically reconfigured and scaled to control the service behavior and performance by acting on the number (and the deployment) of virtual machines.

Combining this capability to quickly change the number of active virtual servers with the flexibility of a composite Web application, service providers can adapt the system to the actual load providing responsive services at reduced costs, as we show in the next Section.

4 Experimental Validation

In this section, we report the results of our experiments in building Kriging models to represent the behaviour of the DoReMap virtualized Web application. The

[4] `http://www.reservoir-fp7.eu/`

experiments aim to verify whether Kriging models can be used as an approximation of the behaviour of a realistic composed system, what kind of SLA-related metrics can be predicted accurately, and how they can be used to choose an optimal system configuration. In more details, the experiments address the following research questions:

Q1 How accurate is the prediction outside the training set?
Q2 How does the quality of the prediction increase with the number of samples?
Q3 How quickly can the model be computed/updated?
Q4 Can surrogate models be used to choose an optimal system configuration?

To answer **Q1** we first build surrogate models using a regular sample set in the feature space, and then we compare their predictions with respect to the system response measured at randomly chosen samples. We address **Q2** by measuring the prediction error of models generated with different sparse and small sample sets with respect to the model computed starting from the full sample set. We evaluate the cost of generating models (**Q3**) by benchmarking our algorithm with increasingly large samples. The speed of the fitting of Kriging models is a critical aspect to determine whether models can be kept up-to-date at run time and thus used to drive the adaptation decisions of an autonomic controller. We use the models to compute the objective function needed for the controller self-configuration functionality (**Q4**).

4.1 Experimental Setup

Our experiments aim to construct surrogate models that represent how different configurations (model input) impact on the system behaviour measured considering different KPIs (model output). As system configurations we consider the set of controllable system parameters (i.e., the number of VMs instantiated per each tier of the Web application) as well as the intensity of the workload (under our control at staging time, but not controllable in production). We measure the workload intensity in terms of number of clients that concurrently access the Web application.

The size of the system configuration space is limited by the available resources on which VMs can be allocated. In our case we deployed our experimental system on 20 physical CPU cores. We allocated the cores as follows: up to 8 cores to run the JOpera (JO) engines, up to 16 cores to run the Doodle Application Servers (AS), one core for the Restaurant Application Server (RS) and one core for the Database Server (DB). The allocation is constrained to at most 20 cores used simultaneously, and to each core dedicated to a single component to avoid contention that would complicate the interpretation of the results. The smallest working configuration thus requires each tier (JO+AS+RS+DB) to be instantiated, for a total of 5 cores, considering that each JOpera engine instance requires 2 cores. The largest configurations that we could test used 4 JOpera instances (consuming 8 cores) with 10 AS instances, and 16 AS instances with one instance of the JOpera engine. We used these configurations to determine the saturation

point of the system (where the throughput stops increasing), and we observed that this occurs with about 100 concurrent clients. Thus, we need to explore 52 possible system configurations for each client workload (from 1 to 100 clients).

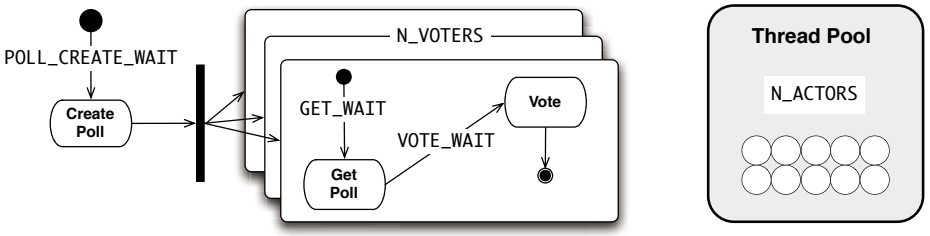


Fig. 5. Workload Model and Workload Generation Parameters

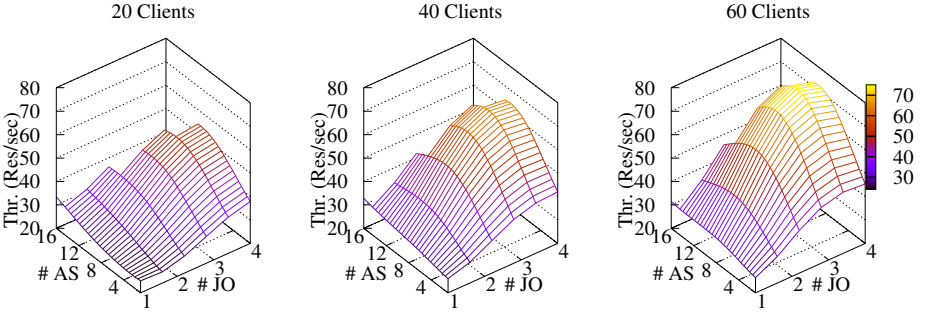
We sampled the parameter space through a batch of controlled experiments executed with Weevil [20]. To minimize undesired randomness, we repeated the experimental runs 5 times per sample, and we measured the output results for a given sample as the average over the run averages. To avoid measuring transient behaviours at component start-up or shut-down, each run lasted 5 minutes, and we discarded the first and last 10 seconds of observation. To stress the system, we used a synthetic client workload generated as a set of Poisson processes with different rates for each request type (`POLL_CREATE_WAIT` = 5 s, `GET_WAIT` = 2 s, `VOTE_WAIT` = 1 s, as shown in Figure 5). We controlled the workload intensity by selecting the number of concurrent client processes (`N_ACTORS`). Due to the specific nature of the Web application, in which the clients must follow a predefined navigation path by following hyperlinks (for instance get request only after poll creation with post, vote request after getting the available options), the effect of adding client processes to the workload is not reflected linearly in the measured system throughput. Rather, an undersized system configuration would result in a slower workload execution.

After collecting system response averages through reproduceable experimental scripts, we computed the Kriging model with the *octgpr*[5] Octave package. We set the parameters of the model training to high error tolerance to smoothly approximate the whole configuration space even with few samples, as shown in the results presented in the next section.

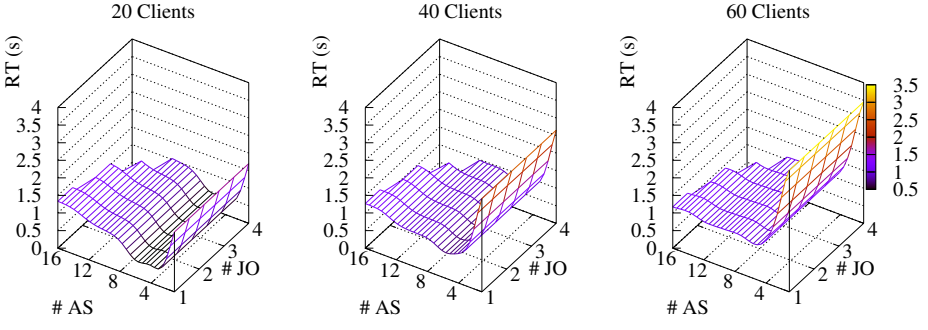
4.2 Results

We sampled and modeled the throughput (Figure 6.a) and the response time (Figure 6.b) that are the most critical KPIs. In both cases we show a projection of the 4-dimensional models by setting the workload intensity to 20, 40, and 60 concurrent clients. The x-y axes show the system configuration in terms of the number of VM instances running the JOpera engine and the Doodle Application

[5] `http://octave.sourceforge.net/octgpr/index.html`



(a) Throughput



(b) Response time of vote request

Fig. 6. Surrogate model as a function of the number of JOpera VMs (# JO), Application Servers VMs (# AS), and Workload (# Clients) built from a regular 100-samples mesh

Server. The z axis shows the predicted throughput (in requests/second) or the response time (seconds).

The model reflects the scalablity of the system, as adding additional resources decreases the response time and increases the throughput. Also, the model predicts that for smaller workloads, the best performance (in terms of response time) is obtained with 7 replicas of the AS tier, while for a larger number of clients, the highest throughput is achieved with 4 JOpera engines and 10 instances of the application server.

Another feature of surrogate models concerns their ability to improve their predictions as more samples are fed into them. To study how the quality of the prediction increases with the number of samples, we built surrogate models using regular sampling patterns of 50, 75, 100 samples, and measured the prediction error with respect to a randomly generated validation set. Table 1 shows that the quality of the prediction increases with the coverage of the parameter space: the mean square error, the root mean square error and the average absolute error significantly decrease as the number of samples used to build the model

Table 1. System throughput: mean square error (MSE), root mean square error (RMSE), average absolute error (ME) and time needed to build the model (Time) with respect to sample sets of increasing size

Training set size	MSE	RMSE	ME	Time
50	117.47	10.838	7.4827	0.0241971 s
75	87.273	9.3420	6.1642	0.048146 s
100	71.402	8.4500	5.2788	0.087447 s

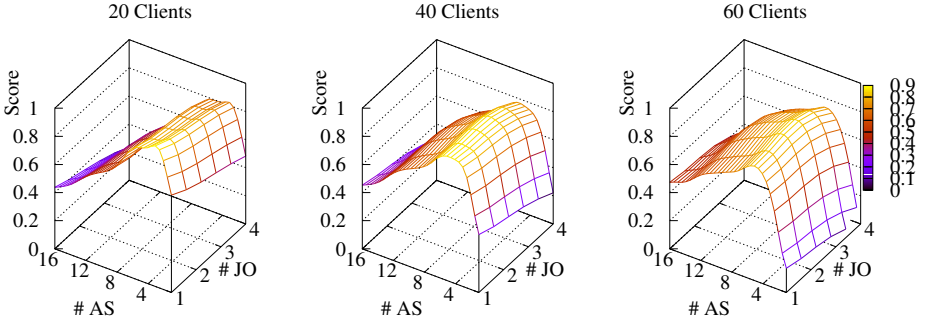


Fig. 7. Aggregated Objective Function (AOF)

increases. The last column of the table shows that the model can be computed in a small amount of time even when using the full set of samples. Thus, these models can be used within the closed control loop envisaged in the architecture of our controller, since they do not introduce a significant delay compared, for example, to the time required to start or shut down a new VM instance.

4.3 Objective Function

Finding a suitable configuration for a system using a set of models like the ones presented in the previous section becomes a multi-objective optimization problem. Several techniques are available to solve the problem, for instance, by ranking objectives (minimize SLA violations, then maximize throughput, and minimize operating costs), or with a single aggregated objective function (AOF), or with Pareto optimization methods.

To show that the surrogate models can be used as suitable input to analyze and optimize configurations, we define an objective function that aggregates them. In this proof of concept, we aggregate the response times for vote requests (R), system throughput (T), and VM operating costs (C) per hour[6] in the following form:

$$AOF(R,T,C) = \alpha * (2s - R) + \beta * T - \gamma * C \quad (1)$$

[6] As cost indication, we used Amazon EC2 Ireland prices mapping AS, DB, and RS to *Small* machines at $0.095 per hour and JO to *Large* at $0.38 per hour.

The AOF translates our service level objectives by giving a negative score for response times above 2 seconds. The second term gives a positive score to high throughputs. This is compensated by subtracting the operating costs of allocating more resources to the system. Parameters α, β, γ let the service designer provide a relative weight for each criterion.

Figure 7 shows the normalized AOF for varying client workloads using values $[\alpha = 10, \beta = 200, \gamma = 5000]$. We observe that the optimal configuration predicted by aggregating the surrogated models into this objective function varies with the number of expected clients respectively to 1 JO and 3 AS for 20 concurrent clients; 2 JO and 5 AS for 40 clients; 3 JO and 6 AS for 60 clients. This results indicates that our approach to modeling the system configuration and its performance can lead to a useful objective function that can be embedded into an autonomic controller.

Additional criteria built from surrogate models can be considered in the AOF (for instance response times for other requests, predicted percentage of SLA violations), as well as other more business-related metrics (for example the difference between the revenue in terms of successfully served requests versus the cost of violations).

5 Related Work

The study and design of controllers for virtualized Web applications is a lively research area. We can give a rough classification of different approaches according to the type of the control technology they adopt (e.g., rule based, control theory) and the knowledge representation that they use (i.e., white-box vs. black-box).

Rule based approaches do not have an explicit representation of the system: domain experts embed their knowledge in event-condition-action rules (ECA) that are evaluated at run time to trigger system adaptation. These controllers have limited capabilities as their effectiveness is bound to the domain experts' ability to define rules, and they do not have built-in learning mechanisms [9].

Control theoretic approaches apply classic control theory and describe system behaviour by means of first principle models or transfer functions. These approaches rely on mathematically-sound control techniques: well known results guarantee the stability of the control under the strong hypothesis of linear system behavior. However, for real systems, model identification (e.g., estimating the transfer function) becomes a difficult and time consuming activity [22]. Basic control theory approaches do not have learning capabilities. Still, more advanced controllers, such as self-tuning regulators (STR), can adapt to the actual system behaviour using different techniques for on-line model parameters estimation [10].

White box feedback loop approaches for controlling virtualized Web applications leverage knowledge of system internals to construct analytical representations in form of Queue Network models: simple *product form* versions of QN that can be analytically solved on-line [1]. Very specialized versions of QNs have been proposed for particular domains, such as multi-tier virtualized Web

applications [4]. Other forms, such as Layered Queue Networks (LQNs), can express more details on system resource contentions and end-to-end behavior [15,5]. Queue networks give reliable system performance predictions and do not require any training of the model. However each change of system configuration requires the entire computation of the model and potentially the re-estimation of all its parameters [17,7].

Black box approaches trade model identification and parameter estimation with feature identification and model training, either in an experimental setting at staging time or through continuous learning while the system is in production. For example, artificial neural networks (ANN) are defined in terms of number of neurons and layers and must be extensively trained before deployment. Once deployed they may need to be retrained if the quality of their predictions decreases [11]. In the cited work, the authors use a ANN to predict if a composite service violates its SLA while continuously retraining the network through monitoring data. A different kind of black box model is employed in [6]. These authors exploits Bayesian Networks (BNs) to predict SLA violations caused by performance problems in a three-tiered application. In the approach, BNs are periodically updated from monitoring data, and the controller can query the model to obtain a probabilistic measure of SLA violation in the near future given the actual working conditions. In general, approaches based on ANNs and BNs are more demanding in terms of samples and training time required to build a reliable model than Kriging [18], hence we deem our solution more appropriate for autonomic controllers for virtualized Web applications where the parameter configuration space is very large and continuous learning is required.

6 Conclusion and Future Work

In this paper we propose to apply Kriging surrogate models to approximate the behavior of a virtualized Web application. This helps systems to automatically and dynamically control how the application is deployed on a cloud infrastructure based on its incoming client workload. We discussed how the main features of Kriging models closely match the requirements of such controllers by providing complete, precise, and quickly update-able representations of the complex multidimensional functions tying system configurations with different performance metrics. We presented our experience in applying our approach to a case study application deployed on a research cloud testbed showing the viability of the approach.

Our current research work concentrates on completing the development and study of a fully functional controller. As a first step, we plan to automate the experiments by actively using the surrogate model error prediction to drive the sampling phase. The second step will be catering for runtime monitoring of the relevant system KPIs and updating the surrogate model accordingly. The final step will be to define suitable optimization policies to make reconfiguration decisions based on aggregated objective functions such as the one presented in this paper. In the long term, we plan to automatically leverage additional

knowledge about the system. For instance, we used the model of the service composition to identify an effective configuration space sampling strategy. That same knowledge can be combined at runtime with surrogate model prediction adopting different strategies to provide the autonomic controller with an improved representation of the system internals.

Acknowledgments

We wish to thank Antonio Carzaniga for the insigtful comments and discussions on the paper. This work is partially supported by the European Community under the IST programme of the 7th FP for RTD - project RESERVOIR contract IST-215605, by the S-Cube NoE and by the Swiss National Science Foundation SOSOA project (SINERGIA grant nr. CRSI22_127386).

References

1. Abrahao, B.D., Almeida, V., Almeida, J.M., Zhang, A., Beyer, D., Safai, F.: Self-adaptive SLA-driven capacity management for internet services. In: Proc. of IFIP/IEEE International Symposium on Integrated Network Management, pp. 557–568 (2006)
2. Almeida, V.A., Menascé, D.A.: Capacity planning: An essential tool for managing web services. IT Professional 4, 33–38 (2002)
3. Brun, Y., Serugendo, G.D.M., Gacek, C., Giese, H., Kienle, H.M., Litoiu, M., Müller, H.A., Pezzè, M., Shaw, M.: Engineering self-adaptive systems through feedback loops. In: Cheng, B.H.C., de Lemos, R., Giese, H., Inverardi, P., Magee, J. (eds.) Software Engineering for Self-Adaptive Systems. LNCS, vol. 5525, pp. 48–70. Springer, Heidelberg (2009)
4. Cunha, I., Almeida, J.M., Almeida, V., Santos, M.: Self-adaptive capacity management for multi-tier virtualized environments. In: Proc. of IFIP/IEEE International Symposium on Integrated Network Management, pp. 129–138 (2007)
5. D'Ambrogio, A., Bocciarelli, P.: A model-driven approach to describe and predict the performance of composite services. In: Proc. of the 6th International Workshop on Software and Performance, pp. 78–89 (2007)
6. Duan, S., Babu, S.: Proactive identification of performance problems. In: Proc. of ACM SIGMOD international conference on Management of data, pp. 766–768 (2006)
7. Ghezzi, C., Tamburrelli, G.: Predicting performance properties for open systems with KAMI. In: Proc. of the International Conference on the Quality of Software Architectures, pp. 70–85 (2009)
8. IBM. An Architectural Blueprint for Autonomic Computing. Technical report, IBM (2003)
9. Jung, G., Joshi, K., Hiltunen, M., Schlichting, R., Pu, C.: Generating adaptation policies for multi-tier applications in consolidated server environments. In: Proc. of International Conference on Autonomic Computing, pp. 23–32 (2008)
10. Karlsson, M., Covell, M.: Dynamic black-box performance model estimation for self-tuning regulators. In: Proc. of the International Conference on Autonomic Computing, pp. 172–182 (2005)

11. Leitner, P., Wetzstein, B., Rosenberg, F., Michlmayr, A., Dustdar, S., Leymann, F.: Runtime prediction of service level agreement violations for composite services. In: Proc. of the Workshop on Non-Functional Properties and SLA Management in Service-Oriented Computing (2009)
12. Lenk, A., Klems, M., Nimis, J., Tai, S., Sandholm, T.: What's inside the cloud? an architectural map of the cloud landscape. In: Proc. of the Workshop on Software Engineering Challenges of Cloud Computing, pp. 23–31 (2009)
13. Pautasso, C.: Composing RESTful services with JOpera. In: Bergel, A., Fabry, J. (eds.) Software Composition. LNCS, vol. 5634, pp. 142–159. Springer, Heidelberg (2009)
14. Pautasso, C., Alonso, G.: The jopera visual composition language. Journal of Visual Languages and Computing 16, 119–152 (2005)
15. Rolia, J., Casale, G., Krishnamurthy, D., Dawson, S., Kraft, S.: Predictive modelling of SAP ERP applications: Challenges and solutions. In: Proc. of the International Workshop on Run-time mOdels for Self-managing Systems and Applications, pp. 2–10 (2009)
16. Sotomayor, B., Keahey, K., Foster, I.: Overhead matters: A model for virtual resource management. In: Proc. of International Workshop on Virtualization Technology in Distributed Computing, pp. 35–42 (2006)
17. Urgaonkar, B., Pacifici, G., Shenoy, P., Spreitzer, M., Tantawi, A.: Analytic modeling of multitier internet applications. ACM Transactions on the Web 1(1), 2–37 (2007)
18. van Beers, W., Kleijnen, J.: Kriging interpolation in simulation: a survey. In: Proc. of Conference on Winter Simulation, pp. 113–121 (2004)
19. Wang, G.G., Shan, S.: Review of metamodeling techniques in support of engineering design optimization. Mechanical Design 129(4), 370–380 (2007)
20. Wang, Y., Rutherford, M.J., Carzaniga, A., Wolf, A.L.: Automating experimentation on distributed testbeds. In: Proc. of International Conference on Automated Software Engineering, pp. 164–173 (2005)
21. Wei, Z., Dejun, J., Pierre, G., Chi, C.-H., van Steen, M.: Service-oriented data denormalization for scalable web applications. In: Proc. of the International Conference on World Wide Web, pp. 267–276 (2008)
22. Zhu, X., Uysal, M., Wang, Z., Singhal, S., Merchant, A., Padala, P., Shin, K.: What does control theory bring to systems research? SIGOPS Oper. Syst. Rev. 43(1), 62–69 (2009)