Iterative Test Suites Refinement for Elastic Computing Systems[*]

Alessio Gambi
University of Lugano, Lugano, Switzerland
alessio.gambi@usi.ch

Antonio Filieri
University of Stuttgart, Stuttgart, Germany
antonio.filieri@informatik.uni-stuttgart.de

Schahram Dustdar
Vienna University of Technology, Vienna, Austria
dustdar@dsg.tuwien.ac.at

ABSTRACT

Elastic computing systems can dynamically scale to continuously and cost-effectively provide their required Quality of Service in face of time-varying workloads, and they are usually implemented in the cloud. Despite their wide-spread adoption by industry, a formal definition of elasticity and suitable procedures for its assessment and verification are still missing. Both academia and industry are trying to adapt established testing procedures for functional and non-functional properties, with limited effectiveness with respect to elasticity. In this paper we propose a new methodology to automatically generate test-suites for testing the elastic properties of systems. Elasticity, plasticity, and oscillations are first formalized through a convenient behavioral abstraction of the elastic system and then used to drive an iterative test suite refinement process. The outcomes of our approach are a test suite tailored to the violation of elasticity properties and a human-readable abstraction of the system behavior to further support diagnosis and fix.

Categories and Subject Descriptors

K.6.4 [**System Management**]: Quality assurance

General Terms

Experimentation, Measurement, Performance

Keywords

Cloud, behavioral modeling, model-based testing

1. INTRODUCTION

Cloud computing is gaining momentum and many companies are moving towards its adoption for business critical applications [16]. On-demand resources allocation and pay-as-you-go cost models are the enabling features for designing elastic computing systems that can adapt to time-varying workloads in order to make applications continuously satisfy Quality of Service (QoS) requirements, still minimizing operative costs.

The main research trends in the area of cloud-based elastic computing are mostly concerned on design principles [14] and benchmarking [17], while quality assurance is often delegated to the dynamic scaling capabilities of the infrastructure only. Supporting elasticity requires effective trade-off strategies. The application is indeed required to scale up quickly in response to a growing incoming workload, though avoiding unnecessary over-provisioning and, furthermore, to efficiently release resources no longer needed (scale down).

A common praxis in designing elastic systems is to tailor the design of resource allocation controllers to specific expected workloads. Being rule-based controllers the state of practice, this usually ends up in the definition of a sub-optimal ruleset to control resources allocation. Most often rules are identified by simulating the application execution under expected workloads (e.g. [11]). The main drawback in such a case is the inability to deal with workload profiles unforeseen at design time. The lack of a convenient methodology in workload profiling for elastic systems exposes designers to the risk of missing the identification of critical usage scenarios that might lead to emergent behaviors and could jeopardize the application execution. This is especially true for cloud computing where classic design commonsense may sometimes fail, as we showed for example in [15], where an elastic system reacted in a quite counterintuitive way to two wave-shaped workloads of different intensity. The prompt reaction of the system to the "high-intensity" wave could misleadingly suggest the system can deal with all the lower intensity loads up to that scale. However, the same system, when solicited with a significantly lower intensity workload, failed to scale correctly. This resulted in an overall degradation of its performance. Islam et al. [17] provided another example where an elastic Web application showed a diverging behavior leading to the acquisition of an ever growing amount of resources from the cloud when subjected to an oscillatory workload with specific, constant, amplitude and oscillation frequency. In both cases, allocation controllers were rule-based.

The previous examples highlight the quest for a paradigm shift in quality assurance methodologies, where elasticity has to be explicitly taken into account.

In this paper we focus on the definition of a systematic model-based test generation framework for the assurance of elasticity properties that iteratively enhances an initial test-

[*]This work is partially supported by the Swiss National Science Foundation under the "Fellowship for Prospective Researches" contract PBTIP2-142337.

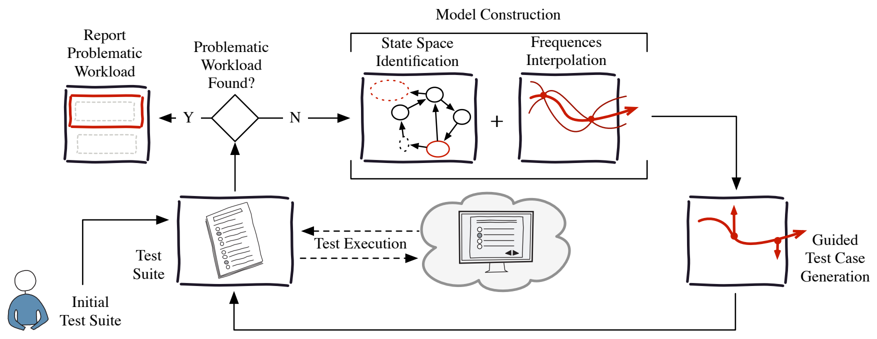
Model Construction
State Space Identification
Frequences Interpolation
Report Problematic Workload
Problematic Workload Found?
Y
N
+
Test Suite
Test Execution
Guided Test Case Generation
Initial Test Suite

Figure 1: Overview of the proposed approach.

suite. Elasticity properties include *elasticity* itself, i.e., the ability of dynamically allocating and deallocating resources on-the-fly, and special forms of its violation including *plasticity*, i.e., the inability to spontaneously return to the original configuration after an adaptation process, and *resonance*, i.e., the emergence of permanent oscillations in resource allocation in response to specific input workloads.

Testing elasticity properties presents several challenges that we try to summarize in the following observations: (i) elasticity depends on complex interactions between computing infrastructures, applications, control logics, and external environmental conditions (e.g., the interaction with third-parties components); (ii) broadly applicable guidelines for testing elastic applications in the cloud are still missing, making handcrafted test cases often biased (as in the previous examples) or unable to capture counterintuitive dynamics; finally, (iii) results obtained by testing an elastic system in a specific cloud environment are hard to generalize to other clouds because of the lack of a wide-scope abstraction model that allows an infrastructure-agnostic representation of elasticity concerns.

The expected contributions of this new research thread can be summarized in the following points:

- an abstract behavioral formalism suitable for capturing and describing elasticity concerns, supporting automatic reasoning about elasticity properties.
- an automatic procedure for inferring systems' behavioral models from test executions and refining them after new test results are gathered.
- an iterative test suite refinement procedure that exploits the acquired knowledge as captured by the abstract model to find test candidates likely to make the system violate its elasticity properties. The execution of the generated test cases produces further information to confirm the presence (respectively, absence) of undesired behaviors of the application, and to enhance the model before the next iteration.

2. THE APPROACH

An overview of the new idea proposed in this paper is provided in Figure 1. The process begins with an initial test suite, either provided by the designer or generated randomly. A test suite contains a set of workload profiles, each of which can be defined as an instantiation of a parametric class of workloads. A parametric class of workload profiles is defined as a time-dependent parametric function $f(\mathbf{w}, t)$, where t denotes the time and $\mathbf{w}$ is a parameters vector, and represents the number of incoming requests at time t.

We require every *feasible* workload to be positive and bounded, i.e., negative or infinite number of requests cannot be issued. Formally, $\exists M \in \mathbb{R}\ s.t.\ 0 \leq f(\cdot,\cdot) < M$, where $\mathbb{R}$ is the set of real numbers. Furthermore, we require every incoming workload to have a *finite horizon*, that is, all the requests have to be issued within a finite time. This implies that $\exists \bar{t} \in \mathbb{R}_+\ s.t.\ \forall t \geq \bar{t}\ f(\cdot, t) = 0$. The second constraint makes it possible to verify whether the system is able to eventually return to its original configuration.

The structure of $f(\cdot,\cdot)$ defines the type of solicitations to test the system with, and can be arbitrarily complex. For example, in [12] a parametric sine function has been used to test the response of the system to oscillatory loads. A workload profile is uniquely characterized within its parametric class by an assignment $\bar{\mathbf{w}}$ of the function parameters. For simplicity, we assume that a test suite is composed by workload profiles belonging to the same parametric class.

The test suite is executed on an instance of the application. If any test case leads to the violation of required elasticity properties, the corresponding workload profile is marked as *problematic* and reported to the designers in order to drive diagnosis and fix. Otherwise, the test suite undergoes a refinement attempt.

The first step for refining the test suite is the construction of a model capturing the elasticity behavior of the system. This model is a Labelled Transition System (LTS) where states capture the current resources allocation and transition labels describe the frequency of transitions occurrence during the execution of the test suite. The initial state of the LTS represents the initial configuration of the system. For the sake of simplicity, here we assume the resources allocation to be simply described by the number of computing instances in use and the controller to be only able to allocate or deallocate identical instances (i.e., not to change their local configuration).

Transition frequency $\phi^{\bar{\mathbf{w}}}_{i,j}$ from state s_i to state s_j is directly computed for each input workload $\bar{\mathbf{w}}$ from test results data as follows. Let $n_{i,j}$ be the number of observed transitions form s_i to s_j during the execution of the workload:

$$\phi^{\bar{\mathbf{w}}}_{i,j} = \frac{n_{i,j}}{\sum_{s_k \in S} n_{i,k}}$$

Intuitively, if the system scales up and down correctly, for all the observed states (possibly with the exception of the initial one) both incoming and outgoing frequencies cannot be zero; otherwise, the system would be stuck in one configuration (plasticity). Furthermore, if the system is behaving properly, to each scale-up should correspond a scale-down, requiring the corresponding frequencies to get close one another. Finally, in case of permanent oscillations (resonance) it would mostly likely be possible to identify parts of the

LTS resembling a strongly connected components. All such situations can be mapped on convenient patterns on transition label values. For the sake of simplicity, in this paper we will focus on plasticity. Analogous approaches can be used for the other mentioned properties.

After computing the transition frequencies, their dependency on the input workload has to be assessed. Being input workloads uniquely characterized by the parameter vector $\bar{\mathbf{w}}$, a set of functions $\hat{\phi}_{i,j}(\mathbf{w})$ are computed in order to approximate the frequencies $\phi_{i,j}^{\bar{\mathbf{w}}}$ by interpolating their dependency on the value of the parameters vector $\bar{\mathbf{w}}$. That is, for a given workload $\bar{\mathbf{w}}$, $\hat{\phi}_{i,j}(\bar{\mathbf{w}}) \approx \phi_{i,j}^{\bar{\mathbf{w}}}$.

The structure of the functions $\hat{\phi}_{i,j}(\cdot)$ is a priori unpredictable. They are in general multivariate (given the size of $\mathbf{w}$) and may be non-linear. Furthermore, being the data generated via random testing it might be the case that the observed frequencies do not constitute a representative sample for inferring the transition functions. For these considerations, parameter-fitting methods may not be suitable for our purpose, driving our choice of using statistical interpolators. The latter are indeed able both to approximate arbitrarily complex functions of the parameter space $\mathbf{w}$ and to provide confidence intervals for each interpolated point. Confidence intervals capture the uncertainty that stems from the limited knowledge due to finite sampling. We are currently planning to achieve statistical interpolation by Gaussian processes [18].

Gaussian processes generalize Gaussian probability distributions in the functional space, providing for each input value $\bar{\mathbf{w}}$ a Gaussian distribution $\mathcal{N}^{\bar{\mathbf{w}}}(\mu, \sigma^2)$ such that, when the number of samples grows, the expected value of the distribution μ converges to the real value of the approximated function and the variance σ^2 tends to 0. For this reason, $\mu(\mathbf{w})$ can be considered as a correct estimator of the interpolated function (in our case any of the frequency functions $\phi_{i,j}^{\mathbf{w}}$), while the variance σ^2 provides a measure of the uncertainty of the estimation. As soon as the interpolation process is completed, a confidence measure on the constructed model can be provided as the maximum variance over $\mathbf{w}$ (or, more precisely, over a reasonable parameter sub-space considered feasible).

Based on the constructed model, the test suite can be improved having in mind two orthogonal goals: identify problematic workload profiles and improve model confidence. The two goals do not interfere one another and can be pursued simultaneously [10].

In order to generate input workloads likely to reveal plasticity of the application, the heuristic we propose is based on finding vectors $\tilde{\mathbf{w}}$ that either make frequencies of self-transitions close to one or frequencies related to scale up (or down) overall close to zero. By exploiting the previously constructed behavioral models, this means finding the values $\tilde{\mathbf{w}}$ that make the proper $\hat{\phi}_{i,j}(\tilde{\mathbf{w}})$ close to one or to zero. Such heuristic may provide a speedup with respect to a random exploration of the parameter space that could be very large even for a small number of parameters. Furthermore, the uncertainty captured by the model provides further guidance to the test case generation process. Indeed, a value $\tilde{\mathbf{w}}$ could make $\mu(\tilde{\mathbf{w}}) \approx 0$ for a transition, but it may turn out to be a false positive due to the inaccuracy of the model. The probability of such erroneous assessment can be bounded by using the variance $\sigma^2(\tilde{\mathbf{w}})$ for a standard hypothesis testing.

The second goal of test case generation is the improvement of the inferred behavioral models. This goal can be achieved by adding test cases corresponding to input regions leading to high values of $\sigma^2(\cdot)$. Adding samples in such regions will improve model confidence and, consequently, make the research of the $\tilde{\mathbf{w}}$ values more effective by reducing the number of false positives.

The new test cases identified by the previous heuristic can then be added to the initial test suite and executed during the next iteration. Notice that the test suite is executed incrementally, thus the execution time of each iteration depends only on the new test cases to be executed.

A possible criterion to decide the termination of the iterative test suite refinement is the reach of an high confidence on the behavioral model while no $\tilde{\mathbf{w}}$ can be found that makes transition function close to zero (or to one for self-transitions).

2.1 Novelty

The work presented in this paper nicely fits along with previous work on queuing theory and performance modeling, and to the best of our knowledge, its contributions that are hereafter summarized have not been explored before:

- The formalization of elasticity properties by means of formal patterns on LTS models.
- The behavioral abstraction strategy accounting for both model structure and transition frequency interpolation as functions of the workload parameters that includes also a confidence measure on the inferred models.
- The model-based heuristic for test case generation that aims both to maximize the probability of generating problematic input workloads and improve the quality of the behavioral model.

Compared to pure random testing, our approach may reduce the number of the generated test cases thanks to the direct search, thus it can reduce the overall testing time and efforts. Furthermore, the confidence measure of the model provides a progress index for the refinement process, and allows to quantify the achieved improvement after each iteration.

3. EXPECTED FEEDBACK

The main points we would like to discuss with the audience are pointed out by the following questions:

- Is the behavioral model that we propose suitable to capture all the elasticity concerns? How about other quality properties that are specific of elastic systems? What would will be a promising alternative to LTS?
- Statistical inference is computationally expensive. A parametric model for frequency functions could allow the use of established and efficient parameter fitting algorithms. Is there any valid parametric model that we can use in place of Gaussian processes?
- How can we exploit the knowledge that we obtained by testing elastic systems on a specific cloud to get insights about their deployment on different platforms? What are the characteristics to take into account in order to assess the validity of reusing previous results, at least up to a certain degree of confidence?
- Is simulation suitable for analyzing elastic applications? Would you recommend specific benchmarks or case studies to assess the validity of our approach?

4. RELATED WORK

In their previous work, the authors investigated the use of Kriging models, a variation of Gaussian processes, for modeling cloud-based systems [13], described the grounding principles of elastic processes, a research agenda for elastic computing and a novel language for their design [4, 5].

The single most related paper by the authors is [12] where they propose an automatic test case generation for elastic systems based on Genetic Algorithms and introduce the idea of parametric workloads for the compact representation of test cases that we used here. In this paper, authors propose an alternative approach for generating test cases that builds an informative model of the system behavior and tries to speed up the generation process following a formally grounded heuristic instead of (pseudo-)randomized search.

Model-based test case generation is a widely investigated topic [1]. Avritzer and Weyuker [2] used behavioral abstractions of software behaviors to drive the automatic generation of load test-suites. In particular, they analyzed a Markov model of the system to identify incoming requests rate likely to degrade its performance. In our approach, workloads are possibly characterized by complex functions, not just their request rate. Analysis methods are consequently different too. More recently, Barna et al. [3] proposed an adaptive method to discover software and hardware bottlenecks and to automatically generate workloads that saturate them. The generation process is guided by performance models fitted from previous runs and drives the system towards the worst case behavior. We follow a similar principle, though we specifically target elasticity properties.

5. OUTLOOK

The approach presented in this paper is undergoing a first evaluation through simulation. The preliminary results obtained for oscillatory workload profiles are encouraging.

After this preliminary assessment, we plan to validate our methodology on real applications. For this purpose, we are developing tools to support trace-based test execution in the cloud and integrating them with popular clouds, such as Amazon EC2 and OpenStack. We will release the prototype in order to let practitioners use it and to collect their feedbacks. We also plan to consider further workload families in order to replicate problematic situations recently reported in literature, thus enabling a thorough comparison.

Among the many aspects that deserve further investigation, we will focus mainly on:

- *Elastic behavior formalization.* The LTS behavioral abstraction provided in this paper leads to finite states automata closely resembling Discrete Time Markov Chains (DTMCs). As a quite intuitive idea, our elasticity properties seem to have tight relations with the notions of ergodicity and recurrence related to DTMCs. If proved correct, such a modeling would pave the way to the adoption of a whole new set of quantitative verification approaches. Also, our heuristic could be enhanced by results from our previous research on parametric model-checking [9, 6], possibly speeding up the search of problematic workloads.
- *Controller abstraction.* In this paper we consider cloud based applications as a whole, including resource allocation controllers and infrastructures. Being able to take into explicit account the controller's logic could provide valuable insights about the dependence of the problematic behaviors not just on the workload but also on specific properties of the resource allocation strategy. This could support the automatic generation of controllers specifically tailored to elastic systems, extending our previous work [7, 8].

6. REFERENCES

[1] S. Anand, E. Burke, T. Y. Chen, J. Clark, M. B. Cohen, W. Grieskamp, M. Harman, M. J. Harrold, and P. McMinn. An orchestrated survey on automated software test case generation. *JSS*, To Appear.

[2] A. Avritzer and E. Weyuker. The automatic generation of load test suites and the assessment of the resulting software. *Software Engineering, IEEE Transactions on*, 21(9):705—716, 1995.

[3] C. Barna, M. Litoiu, and H. Ghanbari. Autonomic load-testing framework. In *Proceedings ICAC*, pages 130–142, 2011.

[4] G. Copil, D. Moldovan, H.-L. Truong, and S. Dustdar. Sybl: an extensible language for controlling elasticity in cloud applications. In *Proceedings of CCGrid*, pages 112–119, 2013.

[5] S. Dustdar, Y. Guo, B. Satzger, and H.-L. Truong. Principles of elastic processes. *IEEE Internet Computing*, 15(5):66–71, Sept 2011.

[6] A. Filieri and C. Ghezzi. Further steps towards efficient runtime verification: Handling probabilistic cost models. In *Proceedings of FormSERA*, pages 2—8, 2012.

[7] A. Filieri, C. Ghezzi, A. Leva, and M. Maggio. Self-adaptive software meets control theory: A preliminary approach supporting reliability requirements. In *Proceedings of ASE*, pages 283—292, Washington, DC, USA, 2011. IEEE Computer Society.

[8] A. Filieri, C. Ghezzi, A. Leva, and M. Maggio. Reliability-driven dynamic binding via feedback control. In *Proceedings of SEAMS*, pages 43—52, 2012.

[9] A. Filieri, C. Ghezzi, and G. Tamburrelli. Run-time efficient probabilistic model checking. In *Proceedings of ICSE*, pages 341—350, 2011.

[10] A. Forrester, A. Keane, and N. Bressloff. Design and analysis of noisy computer experiments. *AIAA Journal*, 44(10):2331–2339, 2006.

[11] S. Frey, F. Fittkau, and W. Hasselbring. Search-based genetic optimization for deployment and reconfiguration of software in the cloud. In *Proceedings of ICSE*, pages 512—521, 2013.

[12] A. Gambi, W. Hummer, and S. Dustdar. Testing elastic systems with surrogate models. In *Proceedings of CMSBSE*, pages 8—11, 2013.

[13] A. Gambi and G. Toffetti. Modeling cloud performance with kriging. In *Proceedings of ICSE*, pages 1439–1440, 2012.

[14] A. Gambi, G. Toffetti, and M. Pezzè. Assurance of self-adaptive controllers for the cloud. In *Assurances for Self-Adaptive Systems*, volume 7740 of *LNCS*, pages 311–339. Springer, 2013.

[15] A. Gambi, G. Toffetti Carughi, C. Pautasso, and M. Pezzè. Kriging controllers for cloud applications. *IEEE Internet Computing*, To Appear.

[16] Gartner Inc. Forecast overview: Public cloud services, worldwide, 2011-2016, 4q12 update. Technical Report G00247462, Gartner Inc., 2013.

[17] S. Islam, K. Lee, A. Fekete, and A. Liu. How a consumer can measure elasticity for cloud platforms. In *Proceedings of ICPE*, pages 85–96, 2012.

[18] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning.* The MIT Press, 2006.