Automated Testing of Cloud-Based Elastic Systems with AUToCLES

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

Waldemar Hummer† and Schahram Dustdar†
† Vienna University of Technology, Vienna, Austria
{lastname}@dsg.tuwien.ac.at

***Abstract*—Cloud-based elastic computing systems dynamically change their resources allocation to provide consistent quality of service and minimal usage of resources in the face of workload fluctuations. As elastic systems are increasingly adopted to implement business critical functions in a cost-efficient way, their reliability is becoming a key concern for developers. Without proper testing, cloud-based systems might fail to provide the required functionalities with the expected service level and costs. Using system testing techniques, developers can expose problems that escaped the previous quality assurance activities and have a last chance to fix bugs before releasing the system in production.**

System testing of cloud-based systems accounts for a series of complex and time demanding activities, from the deployment and configuration of the elastic system, to the execution of synthetic clients, and the collection and persistence of execution data. Furthermore, clouds enable parallel executions of the same elastic system that can reduce the overall test execution time. However, manually managing the concurrent testing of multiple system instances might quickly overwhelm developers' capabilities, and automatic support for test generation, system test execution, and management of execution data is needed.

In this demo we showcase AUToCLES, our tool for automatic testing of cloud-based elastic systems. Given specifications of the test suite and the system under test, AUToCLES implements testing as a service (TaaS): It automatically instantiates the SUT, configures the testing scaffoldings, and automatically executes test suites. If required, AUToCLES can generate new test inputs. Designers can inspect executions both during and after the tests.

I. Introduction and Motivation

The advent of Cloud computing in recent years has deeply changed the way developers think about software application design. Advanced resource allocation and virtualization techniques allow to dynamically acquire resources for computing on demand. One of the often cited key principles for Cloud applications is elasticity [1], [5], which covers two main aspects: 1) scalability, i.e., robust and timely adaptation to cope with workload fluctuations, and 2) (cost-)efficiency, i.e., acquiring only the necessary resources and releasing unutilized resources in an optimized way.

As elasticity is becoming an integral part for applications deployed in the Cloud, systematic testing of elasticity becomes a priority. We collectively denote an elastic application plus the environment it operates in as an *elastic system* (ES). An ES is defined at any point in time by its current elasticity state. For instance, the elasticity state can define the current number of utilized computing machines. Note that the elasticity state in principle can also include more specialized and fine-grained metrics. Take as an example an application that reacts to load fluctuations by actively increasing and decreasing quality of service (QoS) while maintaining the same amount of resources. In this case, the QoS is considered the crucial elasticity property of the system. However, in this work we focus mainly on resource-related elasticity. The elastic behavior causes the ES to switch between elasticity states (denoted *elastic transition*) under certain conditions. A series of such transitions (e.g., a scale-out followed by a scale-in) has been termed *elastic transition sequence* [8] (ETS).

We identify two core motivations for testing elastic Cloud applications. First, it needs to be ensured that, given different patterns of stimuli (inputs), the elasticity behavior of the system corresponds to the expected ETS. The notion of expected ETS can take various forms – some systems may have very detailed specifications (e.g., state transition graph with exact triggering constraints, timing, etc), whereas other systems define elasticity more vaguely (e.g., no more than 10 machines should be used, and the ES should eventually scale back to 1 machine if the load is close to zero). Second, if we take into account that an elastic transition causes a number of non-trivial re-configurations in the system, the correct end-to-end functionality of these ETSs should be thoroughly tested.

Despite its high importance, system testing of cloud-based systems is highly complex and currently lacks appropriate tool support. We identify a number of challenges, including a series of complex and time demanding activities, from deployment and configuration of the ES, to execution of synthetic workloads, to collection and persistence of execution data. Clouds enable parallel executions of the ES, effectively reducing the overall test execution time. However, manually managing the concurrent testing of multiple system instances is tedious and error-prone, hence automatic support for test generation, execution, and management of test data is needed.

A. Motivation for Testing-as-a-Service

We tackle the aforementioned challenges from the perspective of *Testing-as-a-Service* (TaaS) [3] for elastic Cloud applications. In our view, generic TaaS provides major benefits.

First, even though elastic systems come in diverse flavors, the system-level testing concepts are largely generic and can be designed and optimized on an abstract level. For example, elasticity can be exercised by generating time-varying workload distributions which can be easily captured using

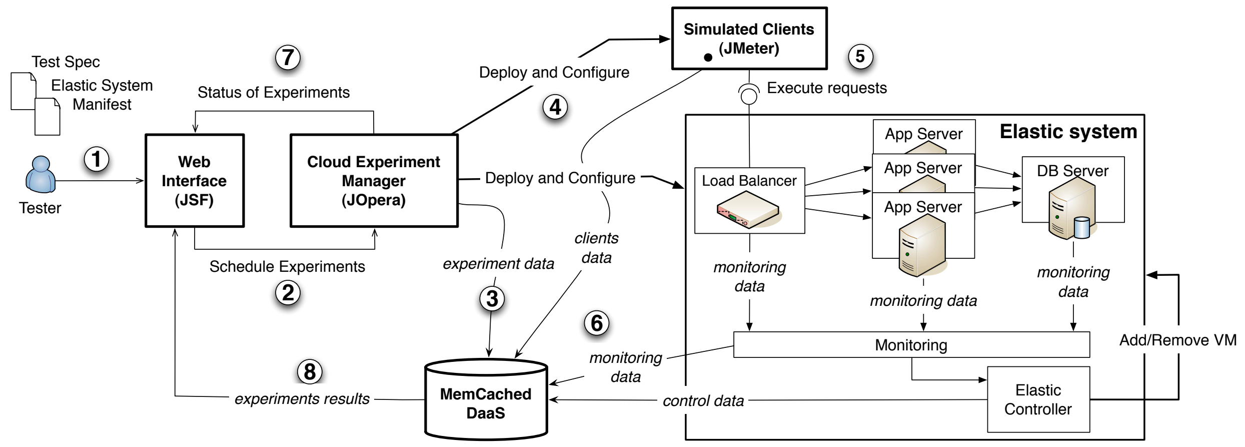

Fig. 1. Overview of AUToCLES

abstracted traces of requests. Traces are then a natural means to support repeatability of tests, enabling fair comparison across implementations. TaaS providers may also elaborate knowledge about tests to create specific test suites for particular ESs.

Second, managing the execution of a multitude of concurrent tests allows the testing platform to apply optimizations with respect to resource utilization and potential consolidation. For example, instead of continuously creating and terminating virtual machines (VMs), the platform can reuse VMs in different runs of the same test suite. Similarly, the platform can share the resources for running synthetic clients.

Third, multiple testers share the Cloud, and hence elastic applications run on the same physical infrastructure. This enables testers to publish test specification and results from past executions to promote fair comparisons among alternative ES implementations. Furthermore, testers and designers may acquire knowledge about the behavior of the cloud under different conditions, thus enabling improvement of their design.

B. Approach Outline

As a first step towards effective and efficient testing of cloud-based elastic systems, we propose AUToCLES (AUtomated Testing Of Cloud-based ELastic Systems), a novel tool that implements *TaaS* for elastic cloud-based applications in the form of black-box system testing.

AUToCLES implements all the core functionalities that are required to automatically manage the whole life cycle of test execution in the cloud, and supports both single-tenant and multi-tenant scenarios by managing independently tests that belong to different users. The tool can run user provided test suites, but it can also be used to create new test suites and evolve them. Thanks to an extensible software architecture and the use of standard technologies, AUToCLES enables an easy integration with existing applications and user-defined testing policies. Furthermore, it adopts a REST-driven architecture that enables scalability, and to some extent elasticity, to achieve cost-effectiveness of the test execution service [9].

AUToCLES captures the repetitive nature of test executions (and their management) in the form of processes that are generally easier to understand, inspect and manage than the commonly adopted set of custom, hand crafted, and distributed batch scripts. At runtime, it augments the elastic systems under test with additional components and virtual machines that provide test scaffoldings and drivers, and deploys the required set of VMs in a specific order. After this, AUToCLES configures the drivers and starts the execution of the test. If the elastic system under test provides a specific monitoring end-point, our tool is able to show to the user the *live*-evolution of the run, and after the run ends, collect any data provided by the elastic system. Data collected from test drivers, i.e., load generators, are automatically collected with no required modification of the elastic system. The acquired testing resources (VMs) are automatically managed and timely released to minimize resource utilization. All collected data are persisted to a data service and can be graphically inspected by the user, with live updates during test execution.

To work correctly the tool assumes that (i) elastic applications under test are specified formally by means of a service manifest, similar to [7], which contains the definition of the various application components, their configuration and their interdependencies; (ii) elastic applications are complete and self-managing, that is, they have all the logic to automatically scale up and down by adding and removing virtual machines at runtime; (iii) the test drivers, i.e., the synthetic clients to generate the load, are available and specified according to well known (*de facto*) standards (see details in Section II).

II. Overview of the Tool

A. Architecture

Figure 1 illustrates the main components of AUToCLES and their relationships along with the elastic systems that we consider in the demo. In our concrete case, the ES under test is a three-tier application controlled by an *elastic controller* that decides on the runtime allocation of resources. The controller should ensure that the system remains elastic, and the purpose of AUToCLES is to assess whether this is done in a reliable manner.

The figure highlights the main steps of the basic scenario that the tool implements, and that we explain in the following[1]: ① The user (tester) uses the Web user interface (UI) to provide a test specification and the elastic system manifest (ESM). The ESM defines the ES in terms of its components and input/output parameters, that is, all information required to automatically and correctly instantiate the ES. Moreover, the ESM lists several KPI (key performance indicator) parameters that are monitored during the execution, along with performance objectives (e.g., feasible value ranges) that should be satisfied at runtime. ② The experiment execution is orchestrated by the light-weight process engine *JOpera* [16] that configures the test clients, and stores all collected experiment data to a *MemCached*[2] data store. ⑤ The test clients are executed using Apache *JMeter*[3], a de-facto standard technology for defining and running load test suites. ⑥ KPI values are continuously monitored and can be accessed through the Web interface at runtime. When the experiments end, all the components persist the monitoring data to the MemCached. ⑦ Along the whole execution, JOpera reports back the status of the experiments that is displayed in the Web UI along with ⑧ the execution data retrieved from the data store. In this way, testers can access and analyze them when convenient.

B. Notes on the Implementation

The general principles behind the implementation are scalability, extensibility and the use of existing standards. Hence, AUToCLES uses several well known technologies (as outlined in Section II-A) to implement a set of core components that can be easily replicated and deployed over a set of distributed nodes. Internally, the tool is organized according to the REST software architectural style, enabling standard HTTP communications between the components and the cloud, in fact reducing the integration and maintenance efforts. Figure 2 illustrates an excerpt of the Web user interface.

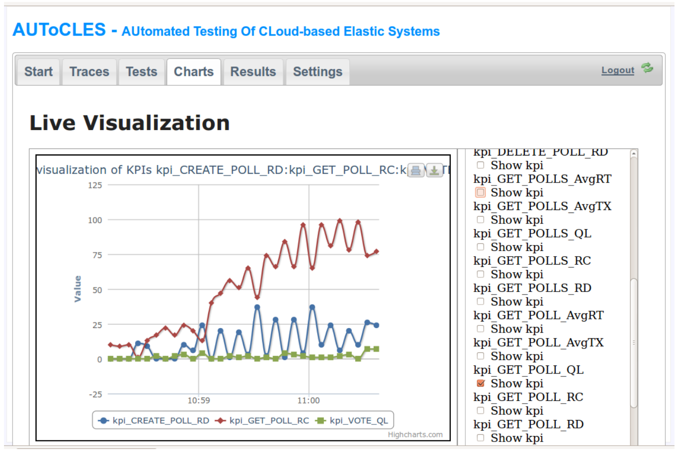

Fig. 2. Excerpt of the AUToCLES User Interface

[1]A video of AUToCLES is available at http://dsg.tuwien.ac.at/autocles
[2]http://memcached.org/
[3]http://jmeter.apache.org/

Managing a set of distributed components instead of a single centralized software may limit the adoption of our tool, therefore we encapsulated all AUToCLES's software components into a single virtual machine that can be run on OpenStack[4] and Amazon EC2[5]. The virtual machine must be run with a specific set of parameters, mainly related to the specific security settings of the cloud. Once the VM is running, no further installation steps are required to start using the tool.

III. Relations with Other Tools

Several tools deal with automated testing of distributed systems, testing automation in clouds, guided generation of tests for exposing problems, benchmarking, and finding optimal system configurations, or combinations of the previous. We are focused on testing elasticity of cloud-based systems, which mainly deals with system testing, therefore, here we generally do not consider tools for automated software testing in the cloud, like, e.g., Cloud9 [4], that exploit the extensive resource availability of clouds to parallelize/distribute software tests. Since AUToCLES depends on reliable provisioning of test machines, existing work on automated testing of infrastructure deployment and configuration [13] complements our approach. Moreover, a number of testing approaches for domain-specific cloud applications are related to our work, for instance testing of elastic platforms for event stream processing [12], [14].

Wang et al. proposed Weevil [18] a tool for automating experimentations over distributed testbeds. Weevil does not specifically target cloud environments, but has a strong focus on managing the distributed aspects of experimentation, and has a noticeable expressive power and flexibility. This increases applicability of the tool, but also forces testers to undergo a steep learning curve, complex setups and difficult test maintenance. We gained experience with Weevil in our previous research [10], [17]. Compared to Weevil, we support a more focused, yet simpler, approach for test automation. Furthermore, we specifically target cloud environments. In this sense, AUToCLES is an easy-to-go solution that fits better with the spirit of clouds in delivering test execution as a service.

Among the tools that are specifically designed for the cloud, we identified ECon [2] and Expertus [15]. ECon was developed by one of the co-authors, shares some software components with AUToCLES, and can be seen as its predecessor. In contrast to AUToCLES, ECon can manage only one experiment at the time and has the main goal to implement adaptive testing, where the next tests are selected combining the data from past executions. Expertus instead automates the experiments in the cloud with the goal of finding the best configuration of the elastic system under test given a target cloud and a set of scenarios. Expertus adopts a generative approach to create on the fly the code that manages the test lifecycle, and to deploy a distributed set of executable scripts. Compared to it, AUToCLES adopt a less intrusive approach to implement test execution and is focused more

[4]http://www.openstack.org/
[5]http://aws.amazon.com/ec2/

on testing a given elastic system than trying to re-optimize its configuration. Expertus points towards design optimization, while AUToCLES focuses more on conformance testing.

In the context of clouds, commercial tools are also available. An outstanding example is Blazemeter[6] that offers load testing as service. Thanks to its capabilities, testers can offload their infrastructure from the burden of generating high amount of load to test their systems. AUToCLES shares with Blazemeter the choice of adopting standard tools and technology for the implementation; in fact, both leverage the DSL defined in the context of the Apache JMeter[7]. Due to this design choice, the designed test plans can be easily executed by both these tools.

IV. Potential Impacts

The original intent of AUToCLES is to support, by means of automatization, efficient and effective system testing of cloud-based elastic systems. This consists of both, consistently managing execution of multiple instances of the system under test, and creation of test suites tailored to testing elasticity.

Nevertheless, the tool has great potential also in broader sense and can provide (or at least, enable the achievement of) several benefits for various actors in the software engineering community. We summarize the most important points:

- AUToCLES enables objective and fair comparisons of different implementations of ESs and control logics. The state of art about the design of ESs, and in general cloud-based, systems is still flawed by the un-availability of shared platform where different actors can repeat the very same experiments to provide a direct comparisons of the results. AUToCLES can be used to delivery a service to access a predefined set of resources outside the control of the testers, and to subject the applications to the very same stress.
- AUToCLES works with multi-tenancy in mind, and might facilitate collaborative approaches for software engineering. For example, it enables sharing of data about test suites and test executions to the different users. These data can be used by designers and practitioners to support different activities that range from the analysis of the quality of system design, to the verification of design-time assumptions about the environment. Collaboration may also be achieved via crowd-sourcing by letting external actors define new test cases, or evaluate the behavior of elastic systems after test execution.
- AUToCLES allows for automated test execution in well-defined, controlled environments, which makes test suites repeatable and comparable. We are currently investigating the tool's potential for regression testing, and we envision that it will prove highly useful for this purpose.
- AUToCLES uses standard technologies and its REST style fosters an easy integration with other tools, for design space optimization, like the SUMO toolkit [11], or test case generation, like the EvoSuite tool [6].

[6]http://blazemeter.com/
[7]http://jmeter.apache.org/

V. Conclusions and Future Work

We discussed the main challenges of testing Cloud-based elastic systems, made the case of automated and optimized test executions, and presented AUToCLES, our tool for automatic execution of trace-based tests for elastic systems in the Cloud.

AUToCLES is designed for extensibility, therefore, new implementations of test case generation and test deployment can be easily used if the currently provided facilities do not perfectly fit specific requirements. In a similar fashion, bindings for using the tool with additional Cloud platforms can be added as well. Moreover, AUToCLES can be easily extended to consider federated (or hybrid) cloud settings to study how elastic systems behave in different environments.

In our ongoing work, we focus on testing and refactoring the code base to improve the quality of the prototype, and on implementing additional strategies for generating test cases, e.g., support for search based and adaptive test-case generation.

Acknowledgements

The work was partially supported by the European Community's Seventh Framework Programme [FP7/2007-2013], grant agreement 257483 (Indenica); the Austrian Science Fund (FWF), grant number P23313-N23 (Audit 4 SOAs); and the Swiss National Science Foundation (SNSF) under the "Fellowship for Prospective Researchers" contract PBTIP2-142337.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, et al. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010.
[2] M. Bisignani. A framework for self-adaptive controllers in the cloud. Master's thesis, Faculty of Informatics Univeristy of Lugano, Feb. 2012.
[3] G. Candea, S. Bucur, and C. Zamfir. Automated software testing as a service. In *ACM Symposium on Cloud computing*, SoCC'10, 2010.
[4] L. Ciortea, C. Zamfir, S. Bucur, V. Chipounov, and G. Candea. Cloud9: a software testing service. *SIGOPS Oper. Syst. Rev.*, 43(4):5–10, 2010.
[5] S. Dustdar, Y. Guo, B. Satzger, and H.-L. Truong. Principles of elastic processes. *Internet Computing, IEEE*, 15(5):66–71, 2011.
[6] G. Fraser and A. Arcuri. Evosuite: automatic test suite generation for object-oriented software. In *ESEC/FSE '11*, pages 416–419, 2011.
[7] F. Galán, A. Sampaio, L. Rodero-Merino, I. Loy, V. Gil, and L. Vaquero. Service specification in cloud environments based on extensions to open standards. In *COMSWARE Conference*, pages 19:1–19:12, 2009.
[8] A. Gambi, W. Hummer, and S. Dustdar. Testing elastic systems with surrogate models. In *CMSBSE Workshop at ICSE'13*, pages 8–11, 2013.
[9] A. Gambi and C. Pautasso. RESTful business process management in the cloud. In *Workshop on Princ. of Eng. Service-Oriented Syst.*, 2013.
[10] A. Gambi, M. Pezzè, and G. Toffetti. Protecting SLA with surrogate models. In *Workshop on Principles of Eng. Service-Oriented Syst.*, 2010.
[11] D. Gorissen, K. Crombecq, I. Couckuyt, T. Dhaene, and P. Demeester. A surrogate modeling and adaptive sampling toolbox for computer based design. *Journal of Machine Learning Research*, 11:2051–2055, 2010.
[12] W. Hummer, O. Raz, O. Shehory, P. Leitner, and S. Dustdar. Testing of data-centric and event-based dynamic service compositions. *STVR*, 2013.
[13] W. Hummer, F. Rosenberg, F. Oliveira, and T. Eilam. Testing idempotence for infrastructure as code. In *Middleware Conference*, 2013.
[14] W. Hummer, B. Satzger, and S. Dustdar. Elastic stream processing in the cloud. *Wiley Interdisciplinary Reviews (WIRE)*, 2013.
[15] D. Jayasinghe, G. Swint, S. Malkowski, J. Li, Q. Wang, J. Park, and C. Pu. Expertus: A generator approach to automate performance testing in iaas clouds. In *Int. Conf. on Cloud Computing*, pages 115–122, 2012.
[16] C. Pautasso and G. Alonso. The JOpera visual composition language. *Journal of Visual Languages & Computing*, 16(1):119–152, 2005.
[17] G. Toffetti, A. Gambi, M. Pezzè, and C. Pautasso. Engineering autonomic controllers for virtualized web applications. In *ICWE'10*, 2010.
[18] Y. Wang, M. J. Rutherford, A. Carzaniga, and A. L. Wolf. Automating experimentation on distributed testbeds. In *ASE*, pages 164–173, 2005.