An Automatic Tool for Benchmark Testing of Cloud Applications

Valentina Casola[1], Alessandra De Benedictis[1], Massimiliano Rak[2] and Umberto Villano[3]
[1]*Università "Federico II" di Napoli, Dipartimento di Ingegneria Elettrica e Tecnologie dell'Informazione, Napoli, Italy*
[2]*University of Campania Luigi Vanvitelli, Dipartimento di Ingegneria dell'Informazione, Aversa, Italy*
[3]*University of Sannio, Department of Engineering, Benevento, Italy*

Keywords: Cloud, Cloud Security, Security SLA, Performance.

Abstract: The performance testing of cloud applications is a challenging research topic, due to the multiplicity of different possibilities to allocate application services to Cloud Service Providers (CSPs). Currently available benchmarks mainly focus on evaluating specific services or infrastructural resources offered by different CSPs, but are not always useful to evaluate complete cloud applications and to discover performance bugs. This paper proposes a methodology to define an evaluation performance process, particularly suited for cloud applications, and an automatic procedure to set up and to execute benchmark tests. The methodology is based on the evaluations of two performance indexes, and is validated by presenting a complete case study application, developed within the FP7-EU-SPECS project. The analysis of the measurement results, produced automatically, can help the developer to discover possible bottlenecks and to take actions to improve both the usability and the performance of a cloud application.

1 INTRODUCTION

The use of virtualized resources hosted in large scale data centers, which is one of the main features of the cloud computing paradigm, has the side effect that the performance of cloud applications, i.e., of the software that runs orchestrating services offered in the cloud, is often very hard to predict. It is well known (Dejun et al., 2009) that infrastructure resources obtained in the cloud may have very different performance behaviors, even if they belong to the same instance type offered by the Cloud Service Provider (CSP). For example, inside the CSP data center, virtual machines (VMs) of the same instance type can be assigned to different physical machines, in different physical locations and under different computing and network load. From the Cloud Service Customer (CSC) point of view, such performance differences are an issue, which may have even an economical impact, due to the pay-per-use paradigm (longer running times may lead to higher leasing costs).

As a consequence, the adoption of benchmarking services, like Cloud Harmony (Gartner, 2017), which regularly execute standard benchmarks and provide customers with average performance figures collected over many different CSPs, is very useful for a coarse-grain comparison of CSPs. However, these services are of little use to predict and to optimize the performance of a specific cloud application.

When the objective is to get insight on the performance behavior of a given application, a possible solution is to perform its *benchmark testing* (Aversano et al., 2013). The results of the application benchmark testing often feed performance prediction tools, based on simulation or analytic models, which make it possible to predict the final effect of multiple configuration and deployment options (Rak et al., 2015; Cuomo et al., 2015; Li et al., 2011; Sharma et al., 2011; Liew and Su, 2012) and to perform effective application performance tuning.

Even if such benchmark-based approach has proven to be fruitful, the complete benchmark testing process can be costly and complex to be executed. Setting up a complete benchmarking procedure for each component of a complex cloud application can be a long and stressing procedure. Moreover, executing the benchmarks and collecting results is both lengthy and expensive, due to the large amount of cloud resources required for the execution in the cloud of the benchmark applications.

In this paper, we propose a technique that makes it possible to automate as much as possible the set-up procedure and the execution of the benchmarks, in order to reduce both the time spent in collect-

Casola, V., Benedictis, A., Rak, M. and Villano, U.
An Automatic Tool for Benchmark Testing of Cloud Applications.
DOI: 10.5220/0006379507290736
In *Proceedings of the 7th International Conference on Cloud Computing and Services Science (CLOSER 2017)*, pages 701-708
ISBN: 978-989-758-243-1

ing the measurements and the complexity of benchmark devising. To demonstrate the feasibility of the approach, we have applied it to evaluate the *SPECS web container* application (SPECS Consortium, 2017d), which is a very complex cloud application that brokers cloud services and automatically configures them according to security Service Level Agreements (SLA), exploiting the SPECS framework (SPECS Consortium, 2017b; Casola et al., 2014).

The reminder of this paper is organized as follows. Section 2 summarizes the methodology adopted to carry out performance tests on a cloud application. Section 3 sketches the structure and high-level behavior of the case study *SPECS web container* application and summarizes the testing conditions and the results collected. Section 4 briefly presents the state of the art and the existing techniques adopted to benchmark and to evaluate the performance of cloud applications, illustrating the main differences compared to the proposed approach. The paper closes with Section 5, where the conclusions are drawn and the future research directions are discussed.

2 THE BENCHMARKING METHODOLOGY

The benchmarking methodology we propose in this paper aims at evaluating the performance of a specific cloud application deployment, as perceived by the application customers, and the overall resource consumption, as resulting from the actual cloud application deployment. Since the cloud computing terminology is still confusing, and even existing standards such as those proposed by ISO (International Organization for Standardization, 2014), NIST (Mell and Grance, 2011), and ETSI (ETSI, 2013) are sometimes contradictory, we briefly introduce here the main terms we will use throughout this paper.

We will use the locution *cloud application* to describe a collection of cooperating software components (simply *components*, in what follows) offered as-a-service. A component can be directly offered as-a-service by a CSP, or by deploying a suitable software artifact over a cloud service of infrastructure capability type (i.e., over a virtual machine).

We will use the locution *cloud application deployment* to identify the mapping of components to resources that leads to a running cloud application by acquiring pre-deployed components provided by a CSP, and/or by deploying custom components over a set of leased virtual machines. As an example, let us consider an application made up of a web application *W* (i.e., a software that offers an HTTP interface)

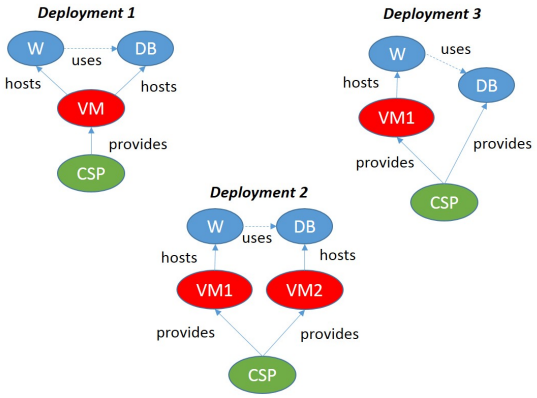

Figure 1: Deployment examples.

which uses a mySQL database *DB*. According to the above-defined terminology, *W* and *DB* are cloud application components. As shown in Figure 1, a possible cloud application deployment can involve a single virtual machine that executes both *W* and *DB*; an alternative deployment requires two virtual machines, hosting *W* and *DB*, respectively; a third alternative can rely on a virtual machine hosting *W*, while the *DB* is offered directly by the CSP (in this case, the number and location of the VM(s) used for running the database service are usually unknown). Further deployment options are linked to the possible tuning of VM parameters (e.g., amount of available memory).

In general, we will assume that a running cloud application has a single access point for its own customers. Moreover, for simplicity's sake we will assume that all components interact through an HTTP interface using RESTful APIs. Hence, in practice, all components can be regarded as web applications. In fact, this assumption allows to exploit an available tool for stressing HTTP-driven components. However, this entails no loss of generality, as the benchmarking method that will be proposed could also be used in different contexts. We are simply interested in identifying the limits in terms of performance of each component of the cloud application. This will make it possible for the application developer to identify possible bottlenecks and to compare different deployments.

In the following subsections, we will discuss the considered performance indexes and illustrate the proposed benchmarking process.

2.1 Performance Indexes

As far as performance indexes are concerned, we can roughly distinguish between two types of cloud applications. The first type has the typical behavior of scientific applications: their execution is launched after

the deployment phase, and ends after the production of results. In the second case, instead, applications are in continuous execution and offer services to customers. The focus of our work is on the second type of applications, which is by far the most common case in the cloud computing context.

According to such considerations, in the following we will consider two main performance indexes:

- **Throughput**: measured in requests/s, it is the number of requests completed per second. In fact, throughput is a measure of the ability of a system to complete the assigned tasks given the amount of available resources.
- **Response Time**: measured in milliseconds (ms), it is the time elapsed between the request of a service up to the end of the service (production of the result). This index is directly related to the performance perceived by end-users, and includes the overhead introduced by the API adopted.

As shown in the next subsection, for each component of the cloud application we will measure an index directly related to throughput (the MAR, defined later) and the response time, in order to discover possible performance-related criticalities. These may be due, for example, to the particular deployment configuration that has been chosen for the application, or even to bugs in its implementation. The analysis of these parameters can help the developer to identify where to operate, in order to improve both the usability and the performance of the application.

2.2 Performance Evaluation Process and Workload Modeling

The performance evaluation process is carried out by injecting suitable workloads for the application, trying to capture the behavior of the application in a real operating environment. In order to automate the performance analysis, we need to build up a set of reusable benchmark tests that can be used to evaluate a generic cloud application deployment. These should suitably model the typical workload the target application and its components are subject to, by means of a set of *synthetic workloads* that can be possibly hardwired in a benchmarking script. As mentioned above, we assume that cloud applications offer their service through dedicated HTTP-based REST APIs exposed by the involved components. As a consequence, all performance tests will stress HTTP layers and will have a request/response behavior.

The *synthetic workloads* have to stress the whole application. In order to do so, we define a set of *load profiles* for each REST API offered by the application

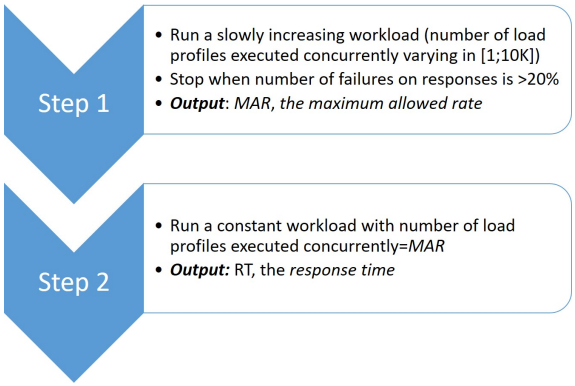

Figure 2: Definition of synthetic workloads.

components. A load profile represents a way to stress the component offering a specific API through the invocation of an API call or through a sequence of API call invocations that characterize the *common* behavior of the component. Each load profile includes all the inputs needed for its execution. Once all the load profiles have been built, each component is stressed in isolation and the performance results collected. This last step can be easily automated through a tool with graphical interface, freeing the application developer from the obligation of boring performance testing sessions and data collection.

As an example, let us consider a component that offers a REST API with two calls, which are commonly called in sequence: the first one retrieves a list of endpoints to objects, and the second one retrieves a single object. A complete service involves the following sequence of requests:

1. Get the list of endpoints to objects.
2. Get one of the objects.

Such sequence of calls constitutes a load profile for the component API, and is a building block for the associated workloads. A *workload* is represented by a set of concurrent executions of a specific load profile, and is characterized by the number of load profiles launched per second.

Figure 2 illustrates the process adopted to define the synthetic workloads. In particular, we first stress the target component with an increasing workload represented by a number of concurrent profile executions per second that grows based on a *ramp function* model (from one to ten thousand concurrent requests, in our experiments), until the limit of correct behavior of the target component is reached. Such limit is reached when the rate of failure[1] for the requests exceeds a given threshold (the default threshold of 20% can be changed through a configuration parameter).

[1]No response or HTTP response 5xx.

The ramp function adopted in this phase has a low slope, to let the workload increase slowly. The request rate that makes the component to reach the failure rate threshold is the *maximum allowed rate* (MAR) of load profiles per second. The actual rate of requests during the normal operation of the component should never exceed the MAR, as it would lead to a severe degradation of performance. So, the MAR should be considered a limit load for the component; the response time measured under this load rate correspond to the worst performance in "acceptable" system behavior. If request rates exceeding the MAR are foreseen in the actual operating conditions of the component, a deployment with multiple instances of the component should be chosen.

In light of the above, it makes sense to measure the response time under a load corresponding to the MAR, which corresponds to the worst operating conditions. So, in the second step, we stress the component with a constant workload represented by a number of concurrent load profiles per second equal to the MAR, submitted continuously to the component for a given period of time. The component *response time* is measured in these conditions.

Following this two-step procedure, we will obtain two values per profile (MAR and response time) for each component. The final performance indexes associated to the component will be the worst values (i.e., the highest response time and the lowest MAR) measured among all tests. As will be discussed in the next section dealing with a complex case study application, the knowledge of these two performance values for each component of the application enables a developer to compare different deployments and to identify the application bottlenecks.

3 CASE STUDY: THE SPECS WEB CONTAINER APPLICATION

In order to validate the proposed methodology, we applied it to evaluate the performance of a complex application developed in the context of the European FP7 project SPECS (SPECS Consortium, 2017b). The application is the *SPECS web container* (SPECS Consortium, 2017d; Casola et al., 2015), available on BitBucket at (SPECS Consortium, 2017c).

The *SPECS web container* application enables a web developer to acquire a cloud-based web container service for web application deployment. The container service ensures the fulfillment of the application developer's security requirements, expressed through a (security) Service Level Agreement (SLA). The web container delivered through the application is built by exploiting the services and tools offered by the SPECS platform. It allows to automatically acquire a web server, which is configured according to security best practices thanks to the adoption of a set of security mechanisms able to grant the requested security SLA.

A comprehensive presentation of the *SPECS web container* application and the SPECS platform is out of the scope of this paper. The interested reader is referred to (Casola et al., 2014) and (Rak et al., 2013) for the related details. However, in the following we will briefly summarize the basic concepts related to the *SPECS web container* as a cloud application, in order to understand how the automated benchmarking process proposed in this paper works.

The SPECS cloud application is made up of five modules, as shown in Figure 3: the *Negotiation*, *Enforcement* and *Monitoring* modules are devoted to manage the corresponding phases of the SLA life-cycle (as defined in the WS-Agreement standard (Andreieux, 2007)), the *SLA Platform* offers the functionalities that enable the inter-operation of the three above modules and to manage the SLA life-cycle, and the *Enabling Platform* is responsible for the execution of all the framework components on a cloud infrastructure.

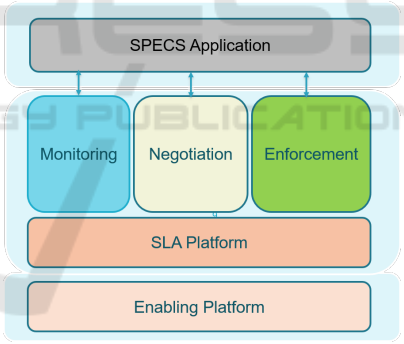

Figure 3: SPECS platform modules.

Each module offers services in the form of REST APIs, and these services are orchestrated by the *SPECS web container* application to provide the *secure* web container to the end-users. The high-level interactions involving the *SPECS application*, the *SLA Platform*, and the *Negotiation* and *Enforcement* modules are sketched in Figure 4 (we neglect here the Monitoring module for simplicity's sake).

The behavior described in the flow represented in Figure 4 is characterized by all the issues discussed in the previous sections, as each module affects both the usage of cloud resources (virtual machines hosting the SPECS modules) and the performance perceived by the end-users. Performing an evaluation of a given SPECS deployment, in order to compare it with other

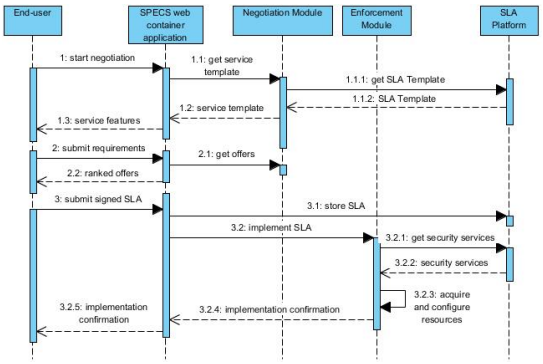

Figure 4: SPECS platform high-level module interactions.

possible distributions of components and/or to minimize the costs in terms of services acquired by a CSP, is a very complex task. In fact, it should be also considered that acquiring two virtual machines of the same type may result in different performance due to their actual distribution on physical resources.

It is worth noting that, independently of the specific behavior of the application under analysis and of its complexity, the methodology proposed in the previous section allows to get insight on the final perceived performance and the load each component is actually able to manage. Moreover, the whole benchmarking process can be easily automated. This can help the developer, for example, to understand if the performance offered to the customer is satisfactory, or whether it is advisable to choose an alternative deployment.

3.1 Performance Testing Environment

To perform the performance testing, we devised a set of scripts to implement the benchmarking procedure described above and to collect its output indexes. These scripts rely on *Gatling* (Gatling Corp, 2017), a load testing tool particularly suited to perform load testing on HTTP servers. Gatling supports the generation of test scripts by means of a *Recorder*, which makes it quite straightforward to build the test sequences (i.e., our *load profiles*) through a graphical interface. The scripts generated for the SPECS platform performance evaluation, representing the *SPECS benchmark*, are available online in the SPECS Bitbucket repository at (SPECS Consortium, 2017a). All scripts have been executed on the SPECS Testbed, a small-sized Eucalyptus cluster that emulates well a typical private cloud or a small CSP.

Figure 5 illustrates the execution environment adopted for the performance tests. It includes (i) the *Testing Terminal Node*, based on the Gatling tool, and the (ii) *SPECS Node*, which includes all the compo-

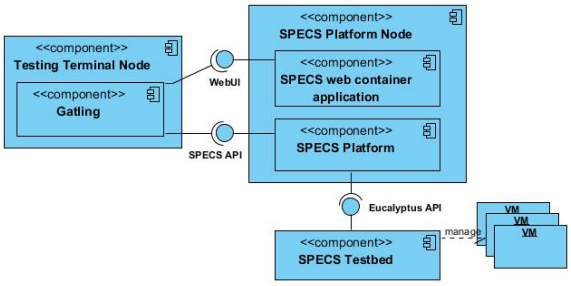

Figure 5: Benchmarking testbed organization.

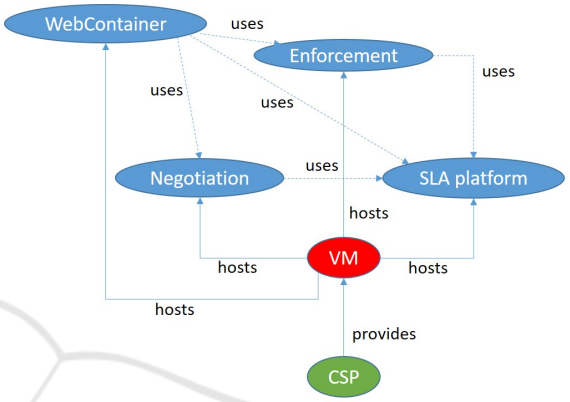

Figure 6: The SPECS cloud application deployment model.

nents of the cloud application under test (the SPECS web container application). As for the cloud application deployment, we assume that all cloud application components (i.e., the SPECS modules and the web container application) run on the same CSP and on a single VM, according to the deployment scenario shown in Figure 6. The *Testing Terminal Node* runs on a separate VM. The features of the VMs used for the testing environment deployment are summarized in Table 1.

Table 1: The VM characteristics.

Node	VM features
Testing Terminal Node	VMtype=1.1xlarge, Core=1, RAM=1024MB, HDD=20GB
SPECS Platform Node	VMtype=1.1xlarge, Core=1, RAM=1024MB, HDD=20GB

The main results obtained for the application are presented in the next section. The complete report of results is available on the Bitbucket repository (SPECS Consortium, 2017a).

3.2 Benchmarking Set-up and Execution

The load profiles for the *SPECS web container* application were built by exploiting the Gatling tool for all components exposed to the end-users (the web-container and its web interface), and manually for the other REST APIs (in fact, even this step could be fully automated). It should also be pointed out that, due to the limitations of the adopted testbed, it was not possible to include in the profiles the steps of the SLA implementation process that involve the acquisition (brokering) of virtual machines.

Figure 7 shows as an example the test results for one of the profiles built for the SLA Platform module (in this case, the load profile stresses the API to get SLAs from the platform). As was to be expected, the throughput grows linearly with the rate of profiles until requests start to fail. From the knee of the curve onward, the throughput still grows, but slowly, due to the high ratio of failure responses. The MAR for the default rate of faults (20%) is reached at about 110 profiles/s. Accordingly, the response time measured stressing the component at MAR for few minutes turns out to be about 95 ms.

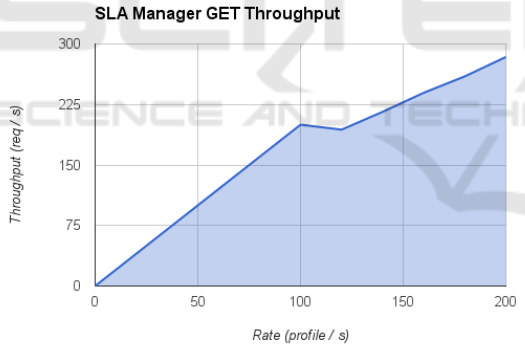

Figure 7: SLA Platform Run Results.

This test was repeated for all the profiles devised for each module. For example, Figure 8 shows the results obtained for a second profile of the SLA Platform module, stressing its API calls for service management. Table 2 summarizes the cumulative results obtained for all the tested modules, reporting only the worst values obtained (our benchmark indexes). According to such results, the application can manage about 3 user requests per second (i.e., about 180 requests per minute). It should be noted that this apparently low value is mostly acceptable for the management of a cluster able to deliver few hundreds of VMs.

The measurements presented above make it possible to discuss how the proposed benchmarking process can be used in practice. The examination of Table 2 induces the suspect of the presence of two bottlenecks, namely in the SLA Platform and in the Web Container module (their MARs are much lower than those of the Negotiation and Enforcement modules). The information of the unexpectedly low performance of the SLA Platform proved to be very useful. It turned out that it was due to a (performance) bug, linked to the use of a *list* method (long queries due to a missing cut on the number of results returned in a single listing). This implementation issue was easily corrected, aligning the SLA Platform module performance to that of the other modules.

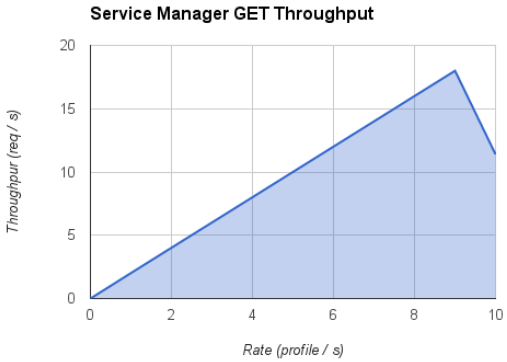

Figure 8: Service Manager Run Results.

Table 2: Single-VM deployment benchmark results.

Component	MAR (reqs/s)	Response Time (ms)
SLA Platform module	9	73
Negotiation module	50	215
Enforcement module	130	180
Web Container module	3	3891

As for the second suspected bottleneck, our successive analyses showed that in the tested deployment the Web Container module, due to its complexity, is intrinsically slower that the other application modules. As these are orchestrated by the "slow" Web Container, they will be never stressed near to their limits. This issue strongly suggests to adopt an alternative different deployment model, where the Web Container can profit from additional resources. For example, it can simply be hosted on a dedicated VM. Due to space constraints, the measurements relative to this second deployment are not reported in this paper.

4 RELATED WORK

As already mentioned in Section 1, the problem of benchmarking cloud services is a well-known open

research topic. A fine-grained performance evaluation of cloud services can enable a cost-reduction policy, but it is hard to perform, due to the elasticity that is a characteristic feature of cloud environments. At the state of the art, most of existing work focuses on infrastructure-as-a-service (IaaS) benchmarking, finalized at the comparison of the infrastructure resources to be leased to execute the applications.

In 2009, the authors of (Binnig et al., 2009) addressed the problem of benchmarking in the cloud by focusing on database services. The paper main goal was to demonstrate the limits of existing benchmarks when applied to the cloud environment. In particular, the paper proposed to adopt new metrics (scalability, cost, peaks, fault tolerance) as indexes for the benchmarks, in order to address the limits of existing solutions.

The work presented in (Li et al., 2010) was one of the first attempts of building cloud benchmarks. The authors proposed a cloud-specific benchmark aimed at comparing performance and cost of different CSPs, called *CloudCmp*. *CloudCmp* collects a set of standard benchmarks and automates their execution, which is carried out in user-space (i.e. the customer acquires the resources and executes the benchmarks).

In order to manage the different choices, (Haak and Menzel, 2011) presented an approach based on the *theory of optimal stopping* that enables an automated search for an optimal infrastructure service as for performance-per-price-ratio, while reducing costs for benchmarking. The performance indexes they considered are network latency and CPU and RAM usage.

The authors of (Scheuner et al., 2014) proposed *Cloud Work Bench* (CWB), a web-service-based tool providing support for the definition of reusable and representative benchmarks, enabling the automatic execution of these benchmarks for comparing IaaS services. In (Scheuner et al., 2015), *Cloud Work Bench* was used to demonstrate the complete cycle of benchmarking an IaaS service with SysBench, a cross-platform standard benchmark suite for the evaluation of several OS parameters.

(Kunde and Mukherjee, 2015) proposed to benchmark the performance of big-data analytics applications by using Hadoop. The paper focused on parameters of interest such as turnaround time and throughput. The main goal was to offer a support for the choice of the infrastructure services best suited for a particular application. For what concerns performance indexes, the paper focused on job execution time and throughput, computed by using a simple workload model based on the *map-reduce* approach.

(Uhlir et al., 2016) proposed a novel methodology for benchmarking CSPs, based on latency measurements collected via active probing, that could be tailored to specific application needs. The benchmark focuses mainly on the latency measured to access the CSPs. The mentioned paper also proposes a procedure to collect and rank the measurements and, accordingly, the CSPs.

From the analysis of the existing literature on cloud benchmarking, two main aspects emerge as fundamental issues, namely the automation of the benchmarking execution (which must be repeatable in order to address cloud unpredictability and elasticity features) and the capability of defining custom benchmarks based on the target application to be run. The approach presented in this paper addresses both aspects. Moreover, at the best of the authors' knowledge, no other currently available solution allows to define dynamically and to execute benchmarks which are both simple (we propose the measurement of just two indexes per application component) and effective to help identify bottlenecks and to compare different deployments for a generic application.

5 CONCLUSIONS AND FUTURE WORK

In this paper, we have illustrated a methodology to devise custom benchmarks aimed at evaluating the performance of a cloud application with different deployment configurations of its components. Under the assumption that the application components offer RESTful APIs and communicate with one another by the HTTP protocol, the benchmarking process can be easily automated and can take advantage of existing testing tools for the generation and submission of workloads. As a matter of fact, a tool with graphical interface has been implemented to drive the tests and to collect the results. The main advantage of the proposed benchmarking methodology is that it relies on the evaluation of just two indexes (MAR and response time) for each component and does not require extensive testing. Such indexes are measured by stressing the components with suitable synthetic workloads that take into account their common usage patterns. As demonstrated for a case study application, the collected results can allow to identify possible bottlenecks, so as to take actions in order to improve performance and usability.

As a future development, we plan to validate our methodology under more complex deployments and by exploiting multiple (public and private) providers. Moreover, inspired by the ongoing work on the MUSA project, we plan to extend the benchmark-

ing methodology to evaluate multiple providers simultaneously, in accordance with the multi-cloud paradigm.

ACKNOWLEDGEMENTS

This research is partially supported by the EC FP7 project SPECS (Grant Agrmt. no. 610795) and H2020 project MUSA (Grant Agrmt. no. 644429).

REFERENCES

Andreieux, A. (2007). Web services agreement specification. https://www.ogf.org/documents/GFD.107.pdf.

Aversano, G., Rak, M., and Villano, U. (2013). The mOSAIC benchmarking framework: Development and execution of custom cloud benchmarks. *Scalable Computing: Practice and Experience*, 14(1).

Binnig, C., Kossmann, D., Kraska, T., and Loesing, S. (2009). How is the weather tomorrow? *Proceedings of the Second International Workshop on Testing Database Systems - DBTest '09*.

Casola, V., A. De Benedictis, Rak, M., and Villano, U. (2014). Preliminary Design of a Platform-as-a-Service to Provide Security in Cloud. In *Proceedings of CLOSER 2014, Barcelona, Spain, April 3-5, 2014*, pages 752–757.

Casola, V., De Benedictis, A., Rak, M., and Villano, U. (2015). SLA-based secure cloud application development: The SPECS framework. In *Proceedings of SYNASC 2015*, pages 337–344.

Cuomo, A., Rak, M., and Villano, U. (2015). Performance prediction of cloud applications through benchmarking and simulation. *International Journal of Computational Science and Engineering*, 11(1):46–55.

Dejun, J., Pierre, G., and Chi, C.-H. (2009). EC2 performance analysis for resource provisioning of service-oriented applications. In *Proceedings of the 2009 Int. Conf. on Service-oriented Computing*, pages 197–207.

ETSI (2013). Cloud standards coordination. Technical report.

Gartner (2017). Cloud Harmony Web Site. https://cloudharmony.com/.

Gatling Corp (2017). Gatling documentation. http://gatling.io/docs/2.2.3/.

Haak, S. and Menzel, M. (2011). Autonomic benchmarking for cloud infrastructures. *Proceedings of the 1st ACM/IEEE workshop on Autonomic computing in economics - ACE '11*, pages 27–32.

International Organization for Standardization (2014). ISO/IEC 17788:2014. Information Technology–Cloud computing–Overview and vocabulary.

Kunde, S. and Mukherjee, T. (2015). Workload characterization model for optimal resource allocation in cloud middleware. *Proceedings of the 30th Annual ACM Symposium on Applied Computing - SAC '15*, pages 442–447.

Li, A., Yang, X., Kandula, S., and Zhang, M. (2010). CloudCmp: Comparing Public Cloud Providers. In *Proceedings of the 10th annual conference on Internet measurement - IMC '10*, pages 1–14.

Li, A., Zong, X., Kandula, S., Yang, X., and Zhang, M. (2011). Cloudprophet: towards application performance prediction in cloud. In *Proceedings of the ACM SIGCOMM 2011 conference*, pages 426–427.

Liew, S. H. and Su, Y.-Y. (2012). Cloudguide: Helping users estimate cloud deployment cost and performance for legacy web applications. In *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th Int. Conf. on*, pages 90–98.

Mell, P. M. and Grance, T. (2011). Sp 800-145. the NIST definition of cloud computing. Technical report, Gaithersburg, MD, United States.

Rak, M., Suri, N., Luna, J., Petcu, D., Casola, V., and Villano, U. (2013). Security as a service using an SLA-based approach via SPECS. In *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th Int. Conf. on*, volume 2, pages 1–6.

Rak, M., Turtur, M., and Villano, U. (2015). Early prediction of the cost of HPC application execution in the cloud. In *Proceedings of SYNASC 2014*, pages 409–416.

Scheuner, J., Cito, J., Leitner, P., and Gall, H. (2015). Cloud workbench: Benchmarking IaaS providers based on Infrastructure-as-Code. In *Proc. of the 24th Int. Conf. on World Wide Web*, pages 239–242.

Scheuner, J., Leitner, P., Cito, J., and Gall, H. (2014). Cloud work bench–infrastructure-as-code based cloud benchmarking. In *Cloud Computing Technology and Science (CloudCom), 2014 IEEE 6th International Conference on*, pages 246–253.

Sharma, U., Shenoy, P., Sahu, S., and Shaikh, A. (2011). Kingfisher: Cost-aware elasticity in the cloud. In *2011 Proceedings IEEE INFOCOM*, pages 206–210.

SPECS Consortium (2017a). SPECS Performance tests. https://bitbucket.org/specs-team/specs-performance-benchmark.

SPECS Consortium (2017b). The SPECS project web site. http://specs-project.eu/.

SPECS Consortium (2017c). The SPECS Secure Web Container Application - Bitbucket Repository. https://bitbucket.org/specs-team/specs-app-webcontainer-rev2.

SPECS Consortium (2017d). The SPECS Secure Web Container Application Description. http://www.specs-project.eu/solutions-portofolio/secure-web-container/.

Uhlir, V., Tomanek, O., and Kencl, L. (2016). Latency-based benchmarking of cloud service providers. In *Proceedings of the 9th Int. Conf, on Utility and Cloud Computing*, UCC '16, pages 263–268.