An Approach for Evaluating Cloud Application Topologies Based on TOSCA

Americo Sampaio, Tiago Rolim, Nabor C. Mendonça, Matheus Cunha
Programa de Pós-Graduação em Informática Aplicada
Universidade de Fortaleza
Fortaleza, Ceará, Brasil
americo.sampaio@unifor.br, tiagorol@gmail.com,
nabor@unifor.br, mathcunha@gmail.com

Abstract— **Provisioning cloud applications usually is a complex task as it involves the deployment and configuration of several components (e.g., load balancer, application server, database) and cloud services (computing, storage, CDN, etc.) also known as application blueprints or topologies. The Topology and Orchestration Specification for Cloud Applications (TOSCA) is a recent standard that has focused on standardizing the way cloud applications are structured and managed to favor interoperability. In this paper we describe an approach that facilitates the evaluation of different application topologies by enabling cloud users to easily provision and evaluate different TOSCA topology options based on performance and cost metrics. We show the technical feasibility of the approach based on a case study with the WordPress blogging application where various topologies were automatically provisioned and evaluated in order to gain insights into the best (w.r.t. performance) options.**

Keywords-Cloud computing; provisioning; topology evaluation, TOSCA

I. INTRODUCTION

Cloud computing has been rapidly adopted by a growing number of organizations that perceive the cloud as a more scalable and cost-effective technical solution as compared to acquiring and maintaining their own local infrastructure [1]. Cloud providers such as Amazon, Rackspace, Google and Microsoft offer a wide range of virtual computing resources that vary in capacity and price as well as range of pricing models (e.g., pay-as-you-go, reserved, spot-based). These resources can be easily acquired/released by the cloud users as their demand grows/shrinks, thus helping them to control more effectively their resource usage and cost. However, it is important to mention that, despite the fact that virtual resources can be easily managed, it is still common that cloud users overprovision their resource selection even when using automated features such as autoscaling [2,3].

Autoscaling cloud services[1] usually focus on adding or removing virtual machines (also referred to as horizontal scaling) based on user-defined thresholds over the consumption of virtual resources such as CPU and memory. Though this can be an effective means to manage many real world applications, specially those with unpredictable demand, many works have highlighted the importance of providing automated ways to provision and test the performance of cloud applications in order to find the best resource types and configuration for particular demand levels [4-7]. An appropriate selection of virtual resources is thus critical for an effective usage of the cloud and is even more important for applications that have a more predictable demand.

Despite of the many benefits offered by existing cloud computing solutions, and the relative simplicity of acquiring and releasing cloud resources, many cloud users still have difficulties in selecting the cloud resources and services that best suit their applications needs [8-10, 21]. For example, previous studies [6,11] have shown that variations in the number and types of the virtual machines (VMs) used to deploy multi-tiered web applications in the Amazon cloud can result in significant performance differences from the application users perspective. In particular, for some specific application layers (e.g., web and application service layer), clustered-based deployment architectures containing multiple low capacity VMs can significantly outperform deployments containing a single higher capacity VM at a much lower cost [6]. Therefore, making the right architectural decisions when deploying applications in the cloud can have a direct impact not only in the applications' operational costs, but also in many other important quality attributes such as performance, scalability, resource usage, security and so on [10,12].

In order to resolve these issues some challenges are involved. Firstly, it is important to have an easy and automated process for deploying the application components (e.g., application servers, load balancers, databases) to be evaluated as well as provisioning resources (e.g. virtual machines, storage, networks) for operating these components [13]. Secondly, it is necessary to have a systematic way of testing the application topology automatically in the cloud environment of the user's choice to investigate the most interesting topology options. From the user´s perspective, it is important that these two challenges are resolved in a single combined fashion and not as separate solutions.

In this paper we present an approach that tackles these two issues in an integrated way. The approach relies on the Topology and Orchestration Specification for Cloud Applications (TOSCA) [14,15] standard for specifying and

[1] https://aws.amazon.com/autoscaling/

2159-6190/16 $31.00 © 2016 IEEE
DOI 10.1109/CLOUD.2016.59

automating the assessment of different application topology options. For example, consider a web-based LAMP stack application such as WordPress, which is composed of an application server and a database server. Our approach enables to specify and automate the deployment, configuration and provisioning of multiple variations of the application topology (e.g., all components in the same VM, application server and database in different VMs, cluster of application servers running in different VMs and so on) using TOSCA as the underlying formalism. Moreover, our approach enables the users to specify different resource types to be tested (e.g., a range of different Amazon instance types combinations to be used for the application components VMs) as well as performance parameters to be measured (e.g. response time). In short, we enable to automate the execution of a variety of performance tests for multiple variations of the application topology in combination with a range of resource types selection. To the best of our knowledge our approach is the first attempt to integrate these two features from a TOSCA based perspective.

The remainder of this paper is described as follows. Section II presents the main concepts and technologies relevant to our work. Section III describes our approach and its current implementation. Section 4 presents the evaluation of the approach through a case study based on the WordPress blogging application. Section 5 compares our approach with related work. Finally, Section 6 concludes the paper and offers suggestions for future work.

II. BACKGROUND

This section presents the TOSCA standard as well as two related technologies that are leveraged by our approach, namely Cloudify [16] and Cloud Crawler [6].

A. TOSCA

Topology and Orchestration Specification for Cloud Applications (*TOSCA*) is a pattern created by OASIS (Organization for the Advancement of Structured Information Standards) to describe cloud application components and their relationships [14, 15]. TOSCA provides a YAML based language, Topology Template (also referred to as the topology model of a service), which is a metamodel for defining applications. The Topology Template defines both the application components as well as how to manage it in a provider-independent way, since significant differences amongst the APIs of different cloud providers are hidden behind the TOSCA implementation.

Figure 1 shows the major elements of a Topology Template: Node Template and Relationship Template. The former describes cloud application components and the latter their relationships. Together, Node Template and Relationship Templates define an application topology as a directed graph. A Node Template specifies the occurrence of a Node Type as a component of a service. A Node Type defines application components properties and the operations available to manipulate these components. This approach allows Node Types to be reused by other Node Templates. Relationship Templates specify the occurrence of a relationship between nodes in a Topology Template. Each Relationship Template refers to a Relationship Type that defines the semantics and properties of the relationship.

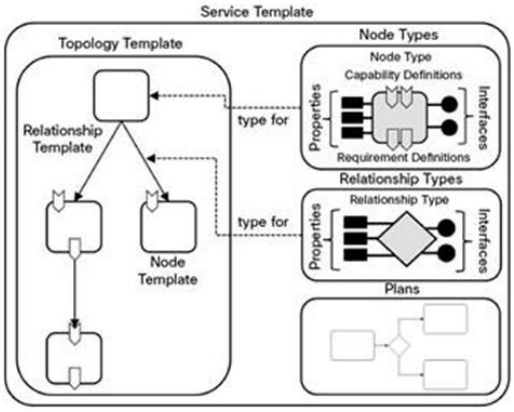

Figure 1. TOSCA service template [14]

B. Cloudify

Cloudify [16] is an open source TOSCA based cloud orchestration software platform implemented in Python. It automates the process of deployment, configuration, and management of the application stack, and was designed to support any application stack and any cloud, which prevents the vendor lock-in. Once Cloudify invokes the infrastructure services offered by different cloud providers, its users can focus on their application. Cloudify users define their application using a TOSCA based domain-specific language (DSL). These definition files, called blueprints, describe the application topology which includes the details required to install and manage the application into the cloud.

C. Cloud Crawler

The *Cloud Crawler* [6] environment supports IaaS cloud users in automatically evaluating the performance of their applications under a variety of topology and workload levels. The environment allows cloud users to specify the performance tests to be executed in the cloud declaratively, at a higher abstraction level. To this end, *Cloud Crawler* provides a declarative domain-specific language, called *Crawl*, which allows the description of a rich variety of performance evaluation application topologies for a given cloud application. *Crawl* also offers a feature for controlling the lifecycle of cloud resources (for example, some virtual machines can be configured to be shut down at the end of each individual test while others can be left running until the last test is executed), thus allowing cloud users to exert a more fine-grained control over their cloud costs. The environment also includes an execution engine, called *Crawler*, which is the service responsible for pre-processing, parsing, validating, executing, monitoring and collecting the results of the performance evaluation topologies described in *Crawl*. *Crawler* is also responsible for configuring and managing the lifecycle of the application components and virtual resources deployed in the cloud, according to the needs and constraints specified for each given topology.

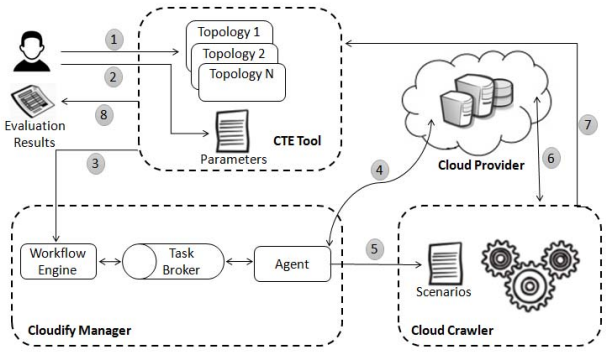

Figure 2. The proposed approach

Cloud Crawler was developed by our own research group and during the tests was responsible for generating the required workload, and deploying, executing and monitoring the performance of the target application. It is important to mention that in the context of this paper Crawler is called by a service from Cloudify after the TOSCA-Based application stack is deployed and configured by Cloudify as detailed in the next section.

III. THE PROPOSED APPROACH

Figure 2 shows the proposed approach in detail. In order to aid understanding of the approach we illustrate the steps based on the Wordpress application used for the evaluation described in section IV. Though the approach is illustrated in this fashion, it can also be used for other types of similar applications based on a multi-tier architecture that could be deployed as various cloud topologies.

In *Step 1* users define a set of topology specifications for that specific application. Each topology specification is a set of files that describe the components of the application as well as its configuration properties. It is important to highlight that the current support of the *Cloud Topology Evaluation tool (CTE)* does not enable to edit the topology specification or generate it from a visual model. These files are created as YAML based on the TOSCA standard and can be reused and extended from previously created files such as Cloudify blueprints. The main idea of *Step 1* is to facilitate the selection of multiple topologies that will later be evaluated by running a set of performance tests configured with parameters provided by the user (e.g., workloads, machine types used, as well as cloud provider parameters) and captured by CTE. In the future we intend to enhance topology creation by using a model-based approach.

Figure 3 shows an example that describes one of the topologies used during our evaluation with Wordpress (wordpress-blueprint1.yaml). This file was originally reused from Cloudify and complemented with the creation of other types as well as parameters provided as input by the user in *Step 2*. Lines 8-12 show input parameters captured by our tool such as image to be utilized, size (in this case an Amazon instance type) and user agent (user that cloudify

```
1  tosca_definitions_version: cloudify_dsl_1_1
2
3  imports:
4    - http://www.getcloudify.org/.../3.2.1/types.yaml
5    - http://www.getcloudify.org/.../1.2.1/plugin.yaml
6    - types/wordpress-types.yaml
7
8  inputs:
9
10   image:
11   size:
12   agent_user:
13
14 node_templates:
15
16   host:
17     type: cloudify.aws.nodes.Instance
18     properties:
19       image_id: { get_input: image }
20       instance type: { get input: size }
21   mysql:
22     type: wordpress.nodes.MySQLDatabase
23     properties:
24       port: 3306
25     relationships:
26       - type: cloudify.relationships.contained_in
27         target: host
28
29   wordpress:
30     type: wordpress.nodes.WordpressApplicationModule
31     properties:
32       port: 80
33     interfaces:
34       cloudify.interfaces.monitoring:
35         start: scripts/crawler/start-monitoring.sh
36     relationships:
37       - type: cloudify.relationships.contained_in
38         target: host
39       - type: node_connected_to_mysql
40         target: mysql
```

Figure 3. Example of one topology file (wordpress-blueprint1.yaml)

```
1  node_types:
2
3    wordpress.nodes.MySQLDatabase:
4      derived_from: cloudify.nodes.DBMS
5      properties:
6        port:
7          description: MySQL port
8          type: integer
9      interfaces:
10       cloudify.interfaces.lifecycle:
11         create: scripts/mysql/install-mysql.sh
12         start: scripts/mysql/start-mysql.sh
13         stop: scripts/mysql/stop-mysql.sh
14
15   wordpress.nodes.WordpressApplicationModule:
16     derived_from: cloudify.nodes.ApplicationModule
17     properties:
18       port:
19         description: Web application port
20         type: integer
21     interfaces:
22       cloudify.interfaces.lifecycle:
23         create: scripts/wordpress/install-wordpress.sh
24         start: scripts/wordpress/start-apache.sh
25         stop: scripts/wordpress/stop-apache.sh
26
27 relationships:
28
29   node_connected_to_mysql:
30     derived_from: cloudify.relationships.connected_to
31     target_interfaces:
32       cloudify.interfaces.relationship_lifecycle:
33         postconfigure: scripts/wordpress/set-mysql-url.sh
34
```

Figure 4. Specific Node Types and Relationships defined for WordPress

utilizes to connect to the VM via SSH). Lines 14-40 define the node templates which represent the components of the application. The host type (lines 16-20) represents an amazon virtual machine whose instance type will be configured based on the input parameter size and image provided previously. This virtual machine will host the MySQL database and the Wordpress application logic (see relationships section defined in lines 26-27; and lines 37-38). For each component we defined a specific node type for the database (line 22) and for Wordpress Business Logic (line 30) as suggested by the TOSCA standard.

Figure 4 shows the node types and relationship types used in the previous topology template. MySQLDatabase (lines 3-13) and WordpressApplicationModule (lines 15-25) are the two node types defined. These types extend previously defined types and define the port properties as well as a set of interfaces that represent phases of the lifecycle (create, start, stop) that contain scripts that are called when these nodes are deployed by Cloudify. Lines 27-33 show the definition of the relationship that connects the wordpress and database. The postconfigure property contains a script that will be called during deployment in order to set the *ip* value of the database in a configuration file of Wordpress.

After completing these definitions, the user triggers the CTE tool to call Cloudify as a service passing the topology files (previous YAML files) as well as the configuration parameters that will later be used during testing with Cloud Crawler. This is what is done in *Step 3*. Inside the Cloudify manager the workflow engine will be responsible for orchestrating the tasks for deployment of the application nodes as well as its configuration (by the task broker). The Cloudify agent is responsible for communicating with the cloud provider API (in this example Amazon AWS) and realizing the deployment and configuration of the whole topology template stack (*Step 4*).

Once the whole application is completely deployed and configured in the cloud, the cloudify tool calls the Cloud Crawler API (*Step 5*) passing the test parameters collected during step one. These parameters such as: list of virtual machine types to be tested; workloads to be executed (e.g., 100, 300, 500, ... concurrent users) and number of rounds of tests are used by crawler to configure its test scenarios. Cloud crawler will then use its load generator to test the application under test (in this case Wordpress) with each of the virtual machine types selected from the list and each workload (*Step 6*). Once the tests are finished, Crawler returns the data collected (*Step 7*) for the CTE tool.

It is important to highlight that the previous steps will be carried on in a repetitive fashion for each topology template to be evaluated. Once all topologies are tested, then the user will have enough data to make a selection of the most appropriate topology configurations for a given workload. This argument will be reinforced in the next session.

IV. EVALUATION

In order to evaluate the usage of our approach we conducted a study to execute automated provisioning and testing of different topologies of the Wordpress application.

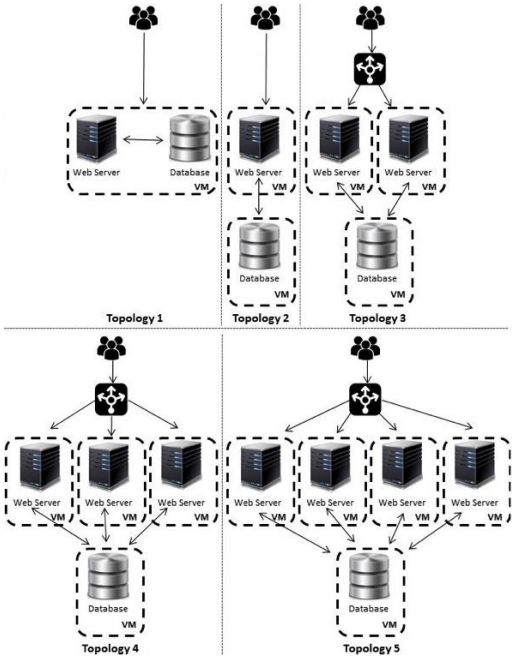

Figure 5. Different topologies for Wordpress

A. Research Questions

This study focused on answering the following research questions:

RQ1. How do different application topologies behave in terms of performance and cost?

RQ2. How do different machine types provisioned affect performance of a given topology?

RQ3. What is the impact of using TOSCA templates for provisioning the different topologies?

B. Study Design

We selected WordPress [11], a real web blogging application, as our application benchmark. This selection was due to WordPress huge popularity, and its scalable architectural components which are ideal for cloud deployment. WordPress follows a typical multi-tiered architecture, comprising two layers: application and data. At each of those layers we deployed the following application components: the Apache application server, and the MySQL relational database server, respectively. To evaluate WordPress performance under varying load conditions we used the Gatling [17] load testing framework. Gatling presents a developer-friendly DSL that allows the definition of a variety of user operations and associated statistics, enabling one to easily test the performance and collect the results of interactive Web applications.

The study considered different variations on the topology of the Wordpress application as show in Figure 5. We basically defined 5 different topologies and provisioned each topology in different Amazon instance types as described below. We defined three different types of instances (M3.Medium, M3.xLarge and M3.large) and apply them to each topology defined and run tests for different workloads

```
node_templates:

  mysql_host:
    type: cloudify.aws.nodes.Instance
    properties:
      image_id: { get_input: image }
      instance_type: { get_input: size }

  mysql:
    type: wordpress.nodes.MySQLDatabase
    properties:
      port: 3306
    relationships:
      - type: cloudify.relationships.contained_in
        target: mysql_host

  wordpress_host:
    type: cloudify.aws.nodes.Instance
    properties:
      image_id: { get_input: image }
      instance_type: { get_input: size_wordpress }

  wordpress:
    type: wordpress.nodes.WordpressApplicationModule
    properties:
      port: 80
    relationships:
      - type: cloudify.relationships.contained_in
        target: wordpress_host
      - type: node_connected_to_mysql
        target: mysql
```

Figure 6. TOSCA specification of topology T2

```
node_templates:

  mysql_host:
    ...

  mysql:
    ...

  wordpress_host:
    type: cloudify.aws.nodes.Instance
    instances:
      deploy: ${instances_deploy}
    ...

  wordpress:
    ...

  nginx_host:
    type: cloudify.aws.nodes.Instance
    properties:
      image_id: { get_input: image }
      instance_type: { get_input: size }

  nginx:
    type: wordpress.nodes.Nginx
    relationships:
      - type: cloudify.relationships.contained_in
        target: nginx_host
      - type: nginx_connected_to_vm
        target: wordpress_host
```

Figure 7. TOSCA specification of topologies T3, T4, T5

(100, 300, 500, 700 and 900 concurrent users). Each test is repeated three times issuing the following sequence of requests: *logon; add a new post; search for the new post; update the new post; search for a previously existing post by keyword; update the existing post; logoff.*

Topology 1 is the simplest containing only one virtual machine that includes the Wordpress logic (Apache server) and the database (MySQL). As described above, this topology will vary the instance type (M3.Medium, M3.xLarge and M3.large) and will be tested 3 times for each of the previous workloads.

Topology 2 separates the web layer (Apache) and MySQL database in two different virtual machines. In this case we will only vary the type of the application server VM (M3.Medium, M3.Large and M3.xLarge) and the database will be fixed as a single instance of type M3.xLarge as recommended by other works [xx]. Again we repeated the tests three times.

Topologies 3, 4 and 5 are based on a slight variation of topology two where the application server (Apache) is replicated respectively in two, three and four VMs and a Load Balancer (Nginx) is added to receive the incoming requests and distribute to one of the Apache servers in the cluster. Tests are also executed three times for the workloads.

C. Study Execution

In order to evaluate the previous topologies we apply the approach defined in section III. We first defined the TOSCA specifications for each of the previous topologies as shown in Figure 6 and Figure 7. Figure 6 shows the definition of topology T2 and Figure 7 shows the specification for topologies T3, T4 and T5. Topology T1 was already shown in Figure 3.

T2 is a variation of the T1 specification where two different hosts are defined as Amazon virtual machines for the MySQL database and the Wordpress application.

T3, T4 and T5 share the same YAML definition where a parameter (*instances)* is configured during deployment time to receive as many number of instances as passed by the user. This will represent the cluster of Apaches used for the Wordpress logic layer.

After describing the topologies, the user provides the input required for the performance tests to the CTE Tool. The parameters passed will be: Cloud credentials; list of machine types to be tested and list of workloads to be executed. These parameters will be used by the approach to configure the tests in Cloud Crawler as discussed in Section III. After all the tests are executed for all topologies with all machine types and workloads the approach is finished and data is analyzed.

D. Results and Discussion

Figure 8 shows the results of the performance tests for the previous topologies.

The data represents the average response times (in milliseconds) for the workloads studied (100, 300, 500, 700, 900) for all topologies except T4 and T5. The reason we omit these two topologies was to improve readability. We evaluate the data with respect to the previous research questions.

RQ1. *How do different application topologies behave in terms of performance and cost?*

In order to assess the data it is also important to highlight that the times correspond to a series of operations and that evaluating what is good or bad will depend on the restrictions of the user and his/her expected quality level. For

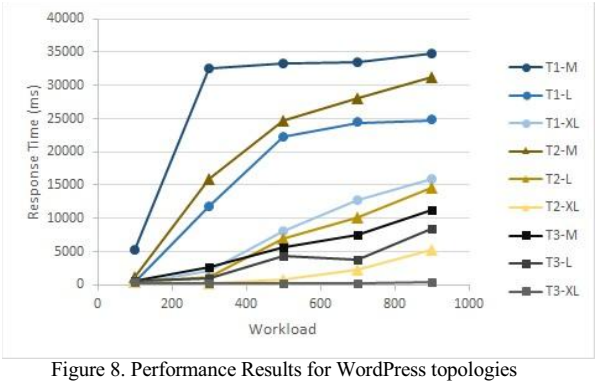

Figure 8. Performance Results for WordPress topologies

this mix of operations we understand, for example, that a response time lower than 10s (10.000ms) is excellent.

Considering the 100 workload for example, one can see that all topologies give very low average response times. The highest would be T1-M (Topology T1 with single virtual machine of type M3-Medium) with approximately 5s that would still be good. T1-M however is not a very reliable configuration as it can be seen that for 300 users its response time increases to approximately 30s on average what can be considered not appropriate by application users.

Separating the database from the application server (T2) shows a great improvement in response times for all machine types selected. For example, for 300 users T2-M gives a response time of about half of the time of T1-M (15s when compared to T1-M around 30s).

Adding a load balancer and application server replicas (T3) improves response times significantly specially for higher workloads. For example the best configuration of T2 (T2-XL) is considerably outperformed by T3-XL (e.g., for 900 users T3-XL averages 403 ms while T2-XL averages about 5000 ms).

This sort of analysis is also very important when considering costs of resource usage. The values by the time of the experiment for each instance type (on demand on West Virginia) are: M3.Medium = $0.067; M3.Large = $0.133; M3.XLarge = $0.266). For example, if users consider the response time of 10 seconds acceptable then the following selection could be applied:

100 users: T1-M would be the cheapest as it uses a single VM of type Medium.

300 users: for this workload two options are the best in terms of cost: T1-XL or T2-L. The first would use a single XL VM of cost $0.266 and the second would use two L VMs giving the same cost. For scalability reasons the user could decide to use the second option.

500 users: for this workload the same reasoning of the 300 workload applies. The only observation here is that in this case T1-XL average response time is closer to the 10s threshold what could reinforce the selection for T2-L.

700 users: in this case T2-M is still the best option in terms of cost but its performance is very close to the threshold. In this case, the T2-XL gives a better performance with average response time of about 2.3s but the cost for the apache VM would double (from $0.133 to $0.266). Another option would be T3.M that would have an intermediate performance (around 7s) but would add another machine for the load balancer and another for the Apache web server.

900 users: For this workload the best choice would either be T2-XL as it would not have the load balancer machine and would have one less Apache virtual machine then T3-L.

RQ2. *How do different machine types provisioned affect performance of a given topology?*

Increasing the size of the virtual machine for any topology (a.k.a vertical scaling) had a direct impact on the results. Practically all topologies tested had an improvement of performance when larger VM sizes were used. However, what the users need to have in mind is that sometimes it is better to increase the number of virtual machine instances (horizontal scaling) then to just increase the VM capacity. For example, topology T3 which is based on a cluster of Apaches significantly outperforms the other topologies especially for higher workloads. Topologies T4 and T5 (not shown in the graph) improve these results even further. Therefore if one needs to have a very strict performance threshold then it is advisable to go for topologies T3, T4 or T5.

RQ3. *What is the impact of using TOSCA templates for provisioning the different topologies?*

The answer for this question is better understood by analyzing Figures 3, 6 and 7. It can be seen that once the user has defined one topology such as topology T1 defined in Figure 3 it is very easy to extend the YAML files for creating the other topologies. In Figure 6 the user only needs to change the previous specification by separating the two components (Wordpress and MySQL) to use two different hosts defined in the YAML specification. For topologies T3, T4 and T5 (Figure 7) the reuse is even better since they can share the same specification file. In this case another host needed to be defined for the NGinx load balancer as well as its node type. A parameter (*instances)* is configured during deployment time to receive as many number of instances as passed by the user. This will represent the cluster of Apaches used for the Wordpress logic layer.

Besides reusing the topology specifications, using TOSCA has the benefit for facilitating the portability of application deployment for other clouds and local infrastructures that are supported by a TOSCA-based implementation such as Cloudify and other tools. In the case of our work that would represent the ability to easily deploy, configure and test the performance of the different topologies for different clouds as Cloud Crawler also supports multicloud deployments [6].

V. RELATED WORK

Automated Provisioning of applications in the cloud is the subject of several works [13, 18, 19, 20]. These works focus on the issue of providing a way for specifying the cloud application topology based on TOSCA [13, 19, 20] or on a proprietary domain specific language [18]. Binz et al. [13] presents a means to combine both flavors to combine declarative and imperative provisioning plans that are generated based on TOSCA topology models. These provisioning plans are workflows that can be executed fully

automatically and can be customized by application developers after generation. Lu et al. [18] implements a deployment service that is similar to the previous approach defining the application's structure through a topology model that is used for provisioning. The approach searches provisioning actions for each component contained in the topology and execute the operations directly without generating an explicit plan that can be modified afterwards. Moreover, this approach is based on proprietary DSL instead of TOSCA what limits its application with respect to portability. Qasha et al. [19] shows how TOSCA can be used to specify the components and life cycle management of scientific workflows by mapping the basic elements of a real workflow onto entities specified by TOSCA. Binz et al. [20] shows the implementation of tool suite for supporting TOSCA packaging, deployment and management. These previous approaches are complementary to our work as they mainly focus on provisioning and do not address performance evaluation of the applications.

Regarding performance evaluation of cloud applications, we can mention several previous works [4, 5, 7] including our own [6]. CloudAdvisor [4] allows users to compare different cloud providers in terms of their estimated price and performance for a given application workload, subject to several user preferences (e.g., maximum budget, throughput expectation, energy saving). The tool requires a detailed characterization of the performance capability of each resource type (e.g., CPU, memory, disk I/O, network) offered by the target cloud platforms, which can be done by executing resource-specific benchmark suites in each cloud. The quality of the estimation will depend on how well the provided resource characterization would match the actual resource needs of the the given application workload.

Expertus is a cloud application evaluation framework that can be used for automatic performance evaluation in IaaS clouds [5]. Expertus employs a template-based code generation approach, which allows the reuse and/or automatic generation of most of the Shell scripts necessary to configure, execute and collect the results of many types of performance tests for a given SaaS application in one or more IaaS cloud providers. According to its authors, Expertus can be quickly customized to accommodate new applications and cloud providers, thus greatly reducing the programming and configuration effort required to run multiple performance tests in the cloud. Unlike Cloud Crawler, Expertus provides automated support for the full performance evaluation cycle, from application deployment to test execution and collection of the results. On the other hand, the fact that Expertus relies on imperative Shell script languages to automate the tests still requires a great deal of lower-level system configuration knowledge from SaaS developers when it comes to specify and execute the types of performance tests they want to conduct with their applications.

CloudBench, in turn, is a cloud experimentation and benchmark service which, like Expertus and Cloud Crawler, can also be used to fully automate a large variety of performance tests in IaaS clouds [7]. Unlike Expertus and Cloud Crawler, CloudBench also supports the execution of scalability tests from the perspective of the cloud provider. The service itself is implemented in Phyton, offering SaaS developers with a rich cloud configuration and test API that can be used either programmatically or by means of a graphical tool. Similarly to Expertus, CloudBench also follows an imperative programming approach, although in a higher abstraction level, based on sequential invocations of the configuration and test operations provided by the service's Python API.

In contrast to both Expertus and CloudBench, Cloud Crawler allows SaaS developers to specify cloud performance tests declaratively, using a flexible YAML based structured language. In this way, it is up to the language interpreter to actually configure, execute and collect the results of the specified test scenarios in the cloud. This declarative approach has the further advantage that it shields the developer from many ``low-level'' cloud deployment and configuration details. However, the limitation of the previous approaches is their lack of support for the TOSCA standard.

Therefore we claim that, to the best of our knowledge, that the work presented in this paper is the best to combine provisioning of TOSCA topologies with the ability to conduct real performance evaluations for different variations on the topology.

VI. Conclusion and Future Work

Selecting an appropriate set of virtual machine types and configurations upon which to deploy a given application is critical to an effective usage of cloud resources. This is particularly true for multi-tiered applications with predictable demand levels, for which existing automated reconfiguration solutions, such as autoscaling, may be hard to fine tune or may result in resource misprovisioning. This paper presented an approach that leverages an existing standard for specifying and deploying cloud application topologies (TOSCA) and existing cloud deployment and performance evaluation tools (Cloudfy and Cloud Crawler, respectively) to facilitate the selection of the best deployment topologies for a given cloud application in terms of a given set of quality attributes. The feasibility and utility of the approach were illustrated by means of a case study conducted with the WordPress blogging application in the Amazon EC2 cloud, where five different WordPress topologies were specified, deployed, tested and analyzed with respect to their operational cost and performance considering different virtual machine types and deployment configurations.

As future work, we plan to extend our approach and support toolset to analyze other application quality attributes, such as security and scalability. We also plan do conduct further experiments to investigate whether and how our approach would generalize to other application domains and cloud platforms, with a special focus on hybrid and multicloud deployment scenarios.

REFERENCES

[1] Michael Armbrust, Armando Fox, Rean Griffith, et al., A view of cloud computing. Communications of ACM, 2010, 53(4), pp. 50-58.

[2] Fabio J. A. Morais, Francisco V. Brasileiro, Raquel V. Lopes, et al., Autoflex: Service Agnostic Auto-scaling Framework for IaaS Deployment Models, CCGRID 2013, pp. 42-49.

[3] Marcelo Gonçalves, Matheus Cunha, Nabor C. Mendonça, Américo Sampaio: Performance Inference: A Novel Approach for Planning the Capacity of IaaS Cloud Applications, CLOUD 2015, pp. 813-820.

[4] G. Jung et al., "CloudAdvisor: A Recommendation-as-a-Service Platform for Cloud Configuration and Pricing," in IEEE SERVICES 2013, pp. 456–463.

[5] D. Jayasinghe et al., "Expertus: A Generator Approach to Automate Performance Testing in IaaS Clouds," in IEEE CLOUD 2012, pp. 73–80.

[6] M. Cunha et al., "A Declarative Environment for Automatic Performance Evaluation in IaaS Clouds," in IEEE CLOUD 2013, 2013, pp. 285–292.

[7] M. Silva et al., "CloudBench: Experiment Automation for Cloud Environments," in IEEE IC2E 2013, 2013, pp. 302–311.

[8] Soren Frey et al. Search-based genetic optimization for deployment and reconfiguration of software in the cloud. ICSE 2013, pp. 512-521.

[9] Saripalli, P., Pingali, G. MADMAC: Multiple Attribute Decision Methodology for Adoption of Clouds. IEEE CLOUD 2011, pp. 316-323.

[10] R. Gonc¸alves Junior et al., A Multi-Criteria Approach for Assessing Cloud Deployment Options Based on Non-Functional Requirements, in ACM SAC 2015, 2015.

[11] Borhani, A. et al. WPress: An Application-Driven Performance Benchmark For Cloud Virtual Machines. IEEE EDOC 2014.

[12] Chung, L. et al. A goal-oriented simulation approach for obtaining good private cloud-based system architectures. J. Syst. Softw, Vol. 86, No. 9, pp. 2242-2262, 2013.

[13] Uwe Breitenbücher, Tobias Binz et al. Combining Declarative and Imperative Cloud Application Provisioning Based on TOSCA. IC2E 2014, pp. 87-96.

[14] OASIS, Topology and Orchestration Specification for Cloud Applications Version 1.0. [Online]. Available: http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html

[15] Topology and Orchestration Specification for Cloud Applications Primer Version 1.0, OASIS. [Online]. Available: http://docs.oasis-open.org/tosca/tosca-primer/v1.0/tosca-primer-v1.0.html

[16] Cloudify website. [Online]. Available: http://www.gigaspaces.com/cloudify-cloud-orchestration/overview

[17] Gatling. Gatling project, stress tool. http://gatling.io/.

[18] H. Lu et al., "Pattern-based deployment service for next generation clouds," in SERVICES, 2013, pp. 464–471.

[19] R. Qasha, J. Cała, P. Watson, Towards automated workflow deployment in the cloud using tosca 8th IEEE International Conference on Cloud Computing, IEEE CLOUD 2015 pp 1037, 1040.

[20] Tobias Binz, Uwe Breitenbücher, Oliver Kopp, Frank Leymann: TOSCA: Portable Automated Deployment and Management of Cloud Applications. Advanced Web Services 2014: 527-549.

[21] PengCheng Xiong, Zhikui Wang, Simon Malkowski, Qingyang Wang, Deepal Jayasinghe, Calton Pu: Economical and Robust Provisioning of N-Tier Cloud Workloads: A Multi-level Control Approach. ICDCS 2011: 571-580.