Elastic-TOSCA: Supporting Elasticity of Cloud Application in TOSCA

Rui Han, Moustafa M. Ghanem, Yike Guo*
Department of Computing
Imperial College London
London, UK
{r.han10, mmg, y.guo}@imperial.ac.uk

Abstract— **The Topology and Orchestration Specification for Cloud Applications (TOSCA) is an emerging framework aiming at enhancing the portability of cloud applications by standardizing their life cycle management in a vendor-neutral way. TOSCA captures the description of cloud application and infrastructure services, the relationships between parts of the services, and the operational behavior of these services (e.g., deploy, patch, shutdown). However, it lacks support for the equally important aspect of managing elasticity, i.e., managing the dynamic scaling of cloud applications at run-time. In this work we present the Elastic-TOSCA framework, which extends TOSCA to address this issue. We then describe how Elastic-TOSCA can be used to support a variety of analytical model-based approaches for elasticity management in complex cloud applications. We further provide a detailed example describing how Elastic TOSCA can be used to support easily a dynamic scaling approach based on a queueing system model. Using a case study for managing the elasticity of a multi-tier e-commerce service, we demonstrate the effectiveness of both the Elastic-TOSCA framework and the scaling approach used.**

Keywords-TOSCA; cloud; elasticity; scaling approaches; queueing system

I. Introduction

Cloud computing has gained unquestionable commercial success in recent years. Key value propositions promoted by cloud IaaS (Infrastructure-as-a-Service) providers such as Amazon AWS (Amazon Web Services) [1] and GoGrid [2] include the user's ability to scale up or down resources used based on their computational demand, thus letting *application owners* (software service creators or developers) pay only for the resources used. This model is appealing for deploying complex applications that provide services for third parties or end users. Some examples of such services include traditional e-commerce sites, online healthcare applications, gaming applications, and media applications. In such applications, if the workload of the service increases (e.g., more end users start submitting requests simultaneously), the application owner ideally needs to scale up the resources used to maintain the Quality of Service (QoS) offered to the end users. When the workload eases down, the application owner ideally needs to scale down the resources used to reduce the cost incurred for service provision. Within this context, supporting dynamic (on-demand) scaling, also known as elasticity, has become one of the most important features that need to be supported in a cloud platform.

The Topology and Orchestration Specification for Cloud Applications (TOSCA) is an emerging framework for describing components' dependencies and deployment plans of cloud applications. Proposed by the Organization for the Advancement of Structured Information Standards (OASIS) [3, 4], TOSCA is designed to simplify the life cycle management of cloud applications in a vendor-neutral way so as to enhance their portability. Such portability is enabled through specifying the operational behaviours of cloud applications, e.g., how servers are deployed or removed and how they are connected, in a uniform way independent of the cloud platform used. This uniform description provides application owners with flexibility when deploying and migrating their applications and associated components across different IaaS providers.

Currently, TOSCA supports the specification of key activities required for the initial deployment of cloud applications and also the activities required to shut down the application. However, it does not provide support for the equally important aspect of specifying how application elasticity can be managed at run-time, e.g., by enabling the specification of how resources can be added or removed at run-time based on workload variation. Our motivation in this paper is to enrich and extend the existing TOSCA framework to support such elasticity management activities in a vendor-neutral way. In particular, our contributions are summarized as follow:

Elastic-TOSCA: We provide extensions to TOSCA that support the specification of dynamic scaling plans and that enable guiding scale-up/down of cloud applications at run-time.

Supporting model-based application scaling approaches using Elastic-TOSCA: The Elastic-TOSCA framework is generic and can support a wide class of dynamic scaling approaches based on analytical models [1, 2, 5-14]. The key requirement for using the framework is that the implemented scaling approach should be able to access its inputs from the Elastic-TOSCA server templates, and to feed its outputs to the template that enables the auto-scaling of applications. We describe and demonstrate how this can be achieved easily using a scaling approach based on a queuing system model [5].

Example implementation and evaluation: We extend the Imperial Smart Scaling engine (iSSe) [10, 15], an intelligent platform designed to automate the deployment and scaling process of cloud applications, to support the Elastic-

* Please direct your enquiries to the communication author Professor Yike Guo.

TOSCA framework. Using iSSe, we evaluate the effectiveness of both Elastic TOSCA and the proposed scaling approach using a multi-tier e-commerce service as an example application.

The remainder of this paper is organized as follows: Section II presents basic concepts on cloud applications and how auto-scaling of such applications can be achieved using analytical model-based approaches. Section III defines the Elastic-TOSCA framework and describes its key components. Section IV describes how a queueing system model-based approach can be supported by Elastic-TOSCA. Section IV introduces the architecture of the iSSe and describes the extensions implemented to support Elastic-TOSCA. It also provides an experimental evaluation of the proposed approach. Finally, Section VI summarises the work presented in this paper and describes avenues for future research.

II. BACKGROUND

In this section, we first describe the structure of a traditional multi-tier application to illustrate how many applications benefit from dynamic scaling in a cloud environment and then discuss existing dynamic scaling techniques that are based on analytical models.

A. *Illustrative Example of Multi-tier Cloud Applications*

The main objective of this paper is to investigate the enrichment of TOSCA to support elasticity management of cloud applications. Addressing this issue effectively requires taking a closer look at the structure of common services and applications that can benefit from dynamic scaling when deployed on IaaS clouds so as to cope with varying workloads. Many such services are typically complex multi-tier applications running on distributed software platforms. Figure 1, shows the logical structure of one such application implemented using four tiers of servers: a frontend HAProxy load balancer for accepting and distributing end users' requests, an Apache web server for handling HTTP requests; a middle-tier Tomcat application server for implementing business logic; and a backend database with data store and processing. These servers work together to handle end users' requests. Depending on the application workload, the servers at each tier can be stressed at different times and the implementation ideally needs to scale up or down the resources at the appropriate tier so as to maintain the overall QoS requirement of the application while minimizing the cost of resources used.

Figure 2(a) shows the lifecycle of the e-commerce application as an example of such dynamic scaling. When the application is initially deployed (see Figure 1), five servers are deployed to support a small number of customers. If the demand increases, the application can be scaled up to add new servers. For example, in the scaling up of Figure 2(b), one Apache server and two Tomcat servers are added to maintain performance. Alternatively, if the demand decreases, some servers can be removed to reduce the cost of service provision. For example, in the scaling down of Figure 2(c), one Tomcat server is removed from Figure 1's initially deployed application.

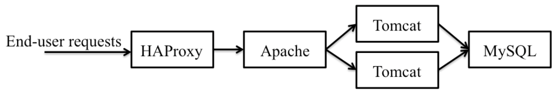


Figure 1. An example multi-tier cloud application.

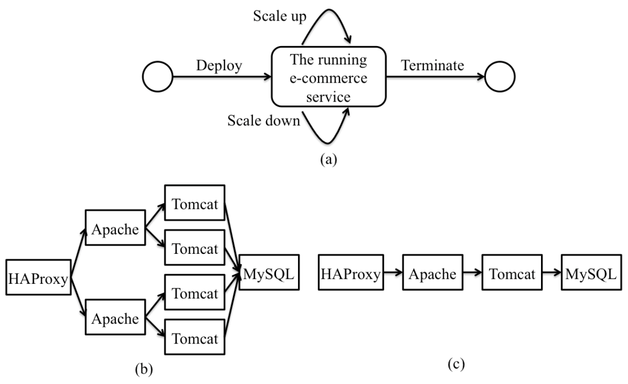


Figure 2. An e-commerce service. We can see (a) the lifecycle of a e-commerce service, (b) the service after a scaling up, and (c) the service after a scaling down.

It should be noted that most scaling approaches for cloud applications, whether used in practice or described in the literature [1, 2, 5-14], are typically based on controlling (increasing or decreasing) the number of Virtual Machine (VM) instances that host the applications' server components. Without loss of generality, we assume that each server component is installed in a stand-alone VM. Accordingly, the scaling up/down the application discussed in this work typically involves adding/removing extra software servers, and hence extra VMs in a cloud environment.

B. *Cloud Scaling Techniques Using Analytical Models*

A variety of approaches that are suitable for auto-scaling multi-tier applications have been proposed in the literature. Many of those employ analytical modelling techniques based on queueing systems [5]. For example, in [6], Xiong et al. model an application by a network of queueing systems and conduct the performance analysis to show relationships among workloads, number of servers and QoS level. In [7], Bacigalupo et al. model an application by a queueing network with three tiers, namely application, database and database disk tiers. Each tier is then solved to analyse the mean response time, throughput and utilisation of a server. In [8], Bi et al. break down an application's end-to-end response time to each tier. They then calculate the number of servers allocated at each tier subject to constraints on the average response time and arrival rate. In [9], Hu et al. consider two allocation strategies using queueing systems: 1) shared allocation (SA) strategy where all incoming requests have the same queueing; 2) dedicated allocation (DA) strategy where requests with different arrival rate are divided into multiple queues. An algorithm is then proposed to decide which strategy (SA or DA) results in a smaller number of servers being used to satisfy the QoS requirement. In addition, in [10], Han et al. consider the cost of VMs and introduce cost-aware criteria to detect and analyse the bottlenecks of multi-tier applications. They then present an adaptive scaling algorithm to lower cost by scaling up or down only at the bottleneck tiers. In [11], Pal et al. propose a pricing framework with economic models designed for multiple cloud providers in the marketplace, where each IaaS provider is modelled as a queueing system. Using this queueing system, the framework aims at informing application owners of the available price and its related QoS level.

Various other stochastic model-based approaches have been studied and used. For example, in [12], Ghosh et al. divide an application into three types of sub analytical model: the resource provisioning decision model, VM provisioning model and run-time model. By iteratively solving each individual sub-model, their analysis obtains two results: response time and service availability. In addition, in [13] Ghosh et al. utilise a stochastic reward net to model an application and provide two analysis results: job rejection rate and response delay. In [14] Li et al. use a network flow model to analyse applications and introduce an approach to assist service providers in making a trade-off between cost and QoS requirements.

It should be noted that the analytical models described here are mainly based on mathematical representations of the application and servers used. Their use in practice requires capturing the structure of the application itself to generate the mathematical representation. Furthermore, their implementation also needs interfacing with the run-time system so as to obtain the parameters used in the models at run-time and also to guide the system in implementing the computed scaling decisions.

III. ELASTIC-TOSCA

In this section, we first introduce the basic TOSCA framework briefly, and then describe how it is extended to define Elastic-TOSCA.

A. Basic Introduction of TOSCA

TOSCA server templates are described in XML and can be used for describing cloud application, including server components and their linking relationships [3, 4]. Figure 3 shows the high-level structure of a TOSCA server template describing an e-commerce service with four sections: Topology template, Node types, Relationship type and Plans. The "Topology template" section specifies the dependency between different server components. The "Node types" section defines the properties of one server, e.g., its owner and the configuration of its hosted VM (CPU numbers, memory size, disk capacity and operating system). A "Relationship type" section specifies the relationship between two servers. In the shown example, an Apache server and a Tomcat server are connected, where the Apache is the source node and the Tomcat is the target node. Finally, the "Plans" section defines the process model for initially deploying a new application and also for removing a running application.

B. Elastic-TOSCA: Extensions to Support Elasticity

We extend the basic TOSCA framework and enrich it with the information required for guiding dynamic scaling of cloud applications, allowing application owners to specify different scaling strategies. For example, an owner could define a scaling up/down strategy based on performance requirements, budgets and QoS requirements specified in service-level agreements (SLAs).

Using the Elastic-TOSCA framework, we generate a new Elastic-TOSCA-based XML document that includes monitoring information structures and new plans for scaling up/down. Figure 4 shows an example server template in Elastic-TOSCA, including two new sections ("Monitoring Information" and "SLA&Constraints") as well as extensions to the "Plans" section, corresponding to three components needed for guiding dynamic scaling of an e-commerce service. Note that the specification and extension of these sections follows TOSCA extensibility mechanism [3, 4], which guarantees that the extended sections are independent of cloud IaaS providers.

The "Monitoring Information" section mainly specifies a running application's current status and underlying infrastructures. In the example fragment in Figure 5, this section records the detected response time and the request arrival rate, as well as the utilisation of resources of the application's hosted VMs.

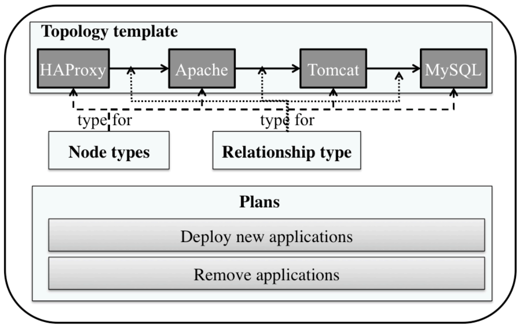


Figure 3. An example server template in basic TOSCA

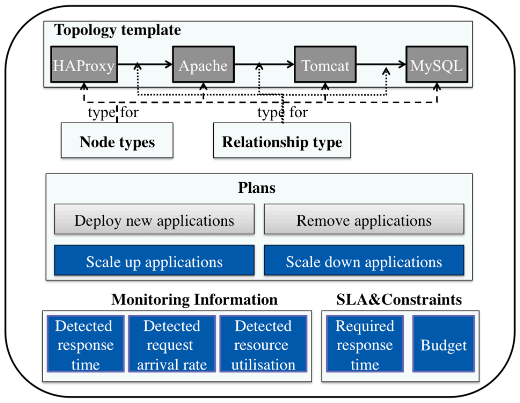


Figure 4. An example server template in Elastic-TOSCA.

```
<MonitoringInformationTemplate id="ApplicationMonitor"
                                name="Application Monitor"
                                InformationType="Monitor">
        <DetectedResponseTime>1.5Seconds</DetectedResponseTime>
        <RequestArrivalRate>100RequestsPerSec</RequestArrivalRate>
        <ResourceUtilisation>
                <CPU>80Percentages</CPU>
                <Memory>75Percentages</Memory>
                <I/O>50Percentages</I/O>
        </ResourceUtilisation>
</MonitoringInformationTemplate>
```

Figure 5. An example “Monitoring Information” section in Elastic-TOSCA.

The “SLA&Constraints” section describes QoS requirements and any constraints on quality, budget, and other aspects of the application. In the example shown in Figure 6, this section specifies the end users’ required QoS: the maximal response time and the application owner’s constraints: the minimal resource utilisation (a resource is considered as idle if its utilisation is smaller than this requirement) and budget (the maximal cost to support the running of the service).

Finally, we extend the “Plan” section in basic TOSCA to define more types of plans that handle the application’s dynamic scaling cases. Figure 4 shows the Elastic-TOSCA definition of two types of scaling plans — “Scale up applications” and “Scale down applications”. Each type can have multiple scaling plans and each plan describes a specific scaling scenario. For example, Figure 7 shows a fragment of a plan for scaling up an e-commerce service. This plan is used for adding one Apache server and two Tomcat servers to the application.

A scaling plan in Elastic-TOSCA server template defines a list of scaling tasks, where each task corresponds to a deployment action. Based on Elastic-TOSCA, this deployment action is independent of any cloud platform, thus enabling applications an IaaS-neutral scaling process. In Figure 8, we provide two example segments in Elastic-TOSCA server templates for specifying a deployment action in two different cloud platforms. These specifications contain all the parameters needed to call an auto scaling API of IC-Cloud [16] (Figure 8(a)) and Amazon AWS [1] (Figure 8(b)) in order to deploy a new Tomcat server when scaling up. Note that for each scaling case, a scaling plan and its scaling tasks are generated dynamically. The information needed to generate documents describing the scaling tasks (e.g., a server’s user name, password and VM configuration) is obtained from the “Node types” section of Elastic-TOSCA.

```
<SLAAndConstraintsTemplate id="ApplicationSLA"
                          name="Application SLA"
                          InformationType="SLA">
        <MaximalResponseTime>2Seconds</MaximalResponseTime>
        <MinimalResourceUtilisation>30Percentages</MinimalResourceUtilisation>
        <Budget>10EUR</Budget>
</SLAAndConstraintsTemplate >
```

Figure 6. An example “SLA&Constraints” section in Elastic-TOSCA.

```
<Plan id=”ScalingUpAnE_CommerceApplication”
        name=”A Scaling Up Plan of E-commerce Website"
        planType="http://docs.oasis-open.org/tosca/ns/2013/01/…."
        languageUsed="http://www.omg.org/spec/BPMN/2.0/">
    <PlanModel>
        <process name="DeployNewApplication" id="p1”>
            <task id="t1" name=”DeployAnApacheServer”/>
            <task id="t2" name="DeployATomcatServer”/>
            <task id="t3" name="DeployATomcatServer”/>
            <sequenceFlow id="s1" targetRef="t2" sourceRef="t1"/>
            <sequenceFlow id="s2" targetRef="t3" sourceRef="t2"/>
        </process>
    </PlanModel>
</Plan>
```

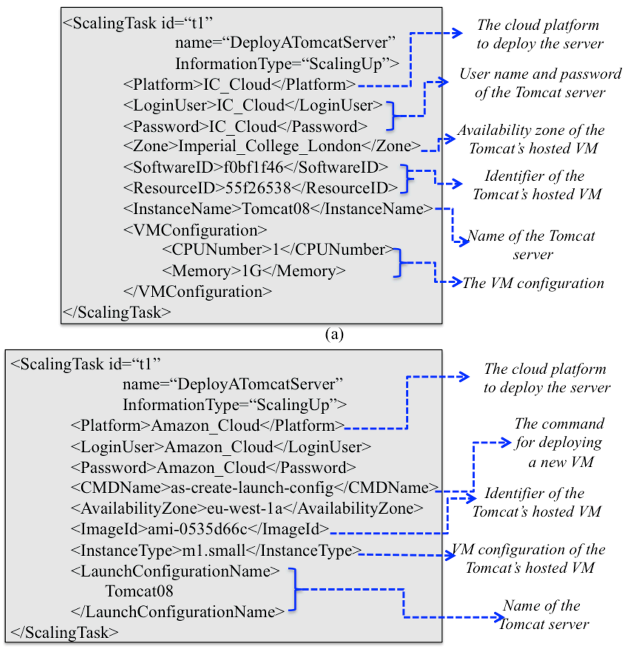
Figure 7. An example scaling up plan in Elastic-TOSCA.

IV. SUPPORTING SCALING APPROACHES BASED ON ANALYTICAL MODELS USING ELASTIC-TOSCA

In this section, we first explain the basic steps of scaling approaches using analytical models and how these steps are supported by Elastic-TOSCA. We then employ a queueing system as a typical example of analytical model to demonstrate these steps.

A. *Analytical Model-based Scaling Approaches Using Elastic-TOSCA*

Typically, an analytical model-based approach for scaling an application consists of four steps, which are preformed using information maintained in different sections of Elastic-TOSCA server templates as illustrate in Figure 9.



```
<ScalingTask id=“t1”
        name=“DeployATomcatServer”
        InformationType=“ScalingUp”>
    <Platform>IC_Cloud</Platform>
    <LoginUser>IC_Cloud</LoginUser>
    <Password>IC_Cloud</Password>
    <Zone>Imperial_College_London</Zone>
    <SoftwareID>f0bf1f46</SoftwareID>
    <ResourceID>55f26538</ResourceID>
    <InstanceName>Tomcat08</InstanceName>
    <VMConfiguration>
        <CPUNumber>1</CPUNumber>
        <Memory>1G</Memory>
    </VMConfiguration>
</ScalingTask>
```

(a)

```
<ScalingTask id=“t1”
        name=“DeployATomcatServer”
        InformationType=“ScalingUp”>
    <Platform>Amazon_Cloud</Platform>
    <LoginUser>Amazon_Cloud</LoginUser>
    <Password>Amazon_Cloud</Password>
    <CMDName>as-create-launch-config</CMDName>
    <AvailabilityZone>eu-west-1a</AvailabilityZone>
    <ImageId>ami-0535d66c</ImageId>
    <InstanceType>m1.small</InstanceType>
    <LaunchConfigurationName>
        Tomcat08
    </LaunchConfigurationName>
</ScalingTask>
```

(b)

The cloud platform to deploy the server
User name and password of the Tomcat server
Availability zone of the Tomcat's hosted VM
Identifier of the Tomcat's hosted VM
Name of the Tomcat server
The VM configuration
The cloud platform to deploy the server
The command for deploying a new VM
Identifier of the Tomcat's hosted VM
VM configuration of the Tomcat's hosted VM
Name of the Tomcat server

Figure 8. Two example scaling tasks for deploying a new server in two cloud platforms: (a) IC-Cloud and (b) Amazon AWS.

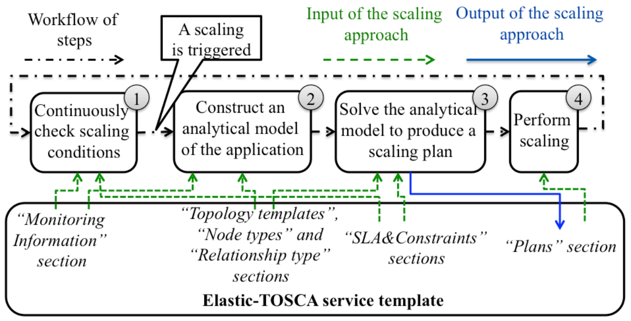

Workflow of steps
A scaling is triggered
Input of the scaling approach
Output of the scaling approach
Continuously check scaling conditions
Construct an analytical model of the application
Solve the analytical model to produce a scaling plan
Perform scaling
"Monitoring Information" section
"Topology templates", "Node types" and "Relationship type" sections
"SLA&Constraints" sections
"Plans" section
Elastic-TOSCA service template

Figure 9. Elastic-TOSCA server templates: for supporting scaling approaches using analytical model.

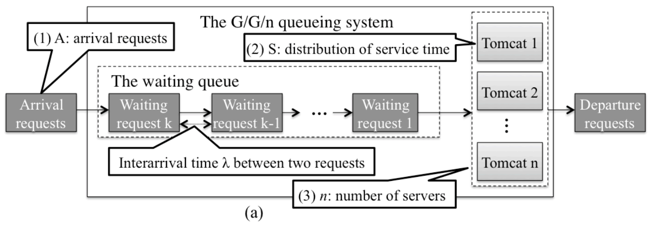
At step 1, the approach continuously checks the "Monitoring Information" and "SLA&Constraints" sections to check where a scaling up/down is needed. The approach proceeds to step 2 if scaling up/down is triggered. At step 2, an analytical model is constructed according to the application topology, configurations of servers and their linking relationships. At step 3, the approach employs analytical modelling techniques to transform the high level QoS requirements into the number of servers to be deployed, and generates a scaling plan for meeting QoS requirements. Finally, step 4 executes the scaling plan.

B. Basic Introduction to Queueing Systems

Typically, a queueing system can be described using $A/S/n$, where A represents the arrival process, S represents the distribution of service time and n is the number of servers [5]. In the example queueing system of Figure 10(a), $A/S/n = G/G/n$ (G for general). This $G/G/n$ queueing system includes n parallel and independent servers and one request waiting queue, where both requests' interarrival time λ (the reciprocal of requests' arrival rate) and servers' service time follow arbitrary distributions. The $G/G/n$ queueing system of Figure 10(a) is used to model a tier of Tomcat servers. Furthermore, the whole m-tier application is modelled as a network of m $G/G/n$ queueing systems and each queueing system represents a tier in the application. Take Figure 10(b)'s 4-tier e-commerce service as an example, which is modelled by a queueing network of four $G/G/n$ queueing systems. The queueing system of the first tier (HAProxy servers) only receives requests from end users, and the departure requests of one tier are the incoming requests of its following tier.

C. Scaling Approach Using Queueing Systems

Queueing theory [5] has been successfully applied in many cloud scaling algorithms [5-11] to perform capacity planning for dynamic scaling. Typically, the overall scaling approach can be described in Figure 11's pseudocode. This scaling approach described in this pseudocode corresponds to Figure 9's four generic steps:



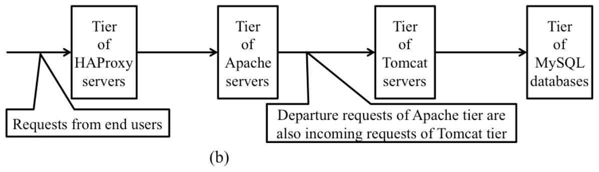


Figure 10. Queueing model and neiworks. We can see (a) a G/G/n queueing system to describe a tier of Tomcat servers, and (b) a queueing network to describe a e-commerce service.

At step 1 (line 3 and 4), the approach decides whether to trigger a scaling up/down according to the latest monitoring information of the running application maintained in the "Monitoring Information" section of Elastic-TOSCA server templates and QoS requirements and constraints to be satisfied in the "SLA&Constraints" Section. For example, if the monitored response time (e.g., 3 seconds) exceeds the maximal required response time (e.g., 2 second), a scaling up is triggered. In contrast, if the detected resource utilisation (e.g., 20%) is below the minimal resource utilisation (e.g., 30%), a scaling down is conducted to remove some idle servers.

At step 2 (line 6 and 10) and step 3 (line 7 and 11), the approach applies a queueing network to model the application and generate a scaling up or down plan (line 7 and 11). This plan is then added to the "Plan" section of Elastic-TOSCA server templates. Section IV.D explains these two steps in detail.

Finally, at step 4 (line 8 and 12), the approach performs the scaling according to the generated plan.

```
A dynamic scaling approach using queueing systems
Input: A m-tier application, QoS requirements and constrains.
1.  Begin
2.    while (the application is not terminated)
3.    Monitor the application's running status and its underlying infrastructures
4.        Check the QoS requirements and constraints to be satisfied
5.        if a scaling up condition is met, then
6.            Construct a queueing network as the analytical model
7.            Conduct capacity estimation to generate a scaling up plan
8.            Perform the scaling up to add servers
9.        else if a scaling down condition is met, then
10.           Construct a queueing network as the analytical model
11.           Conduct the capacity estimation to generate a scaling down plan
12.           Perform scaling down to remove servers
13. End
```

Figure 11. Pseudocode of a dynamic scaling approach using queueing systems.

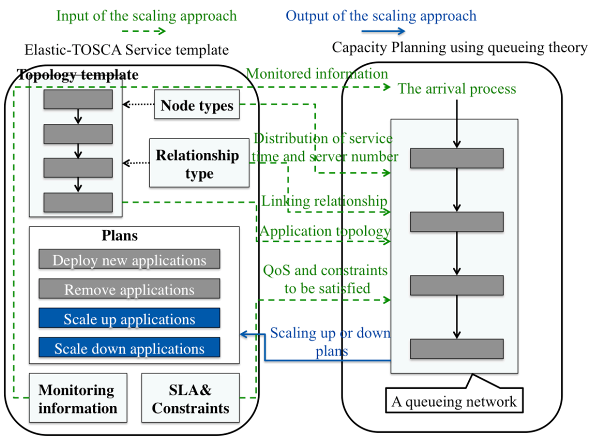


Figure 12. The capacity planning using server templates in Elastic-TOSCA and queueing systems.

D. *Supporting the Scaling Approach using Elastic-TOSCA server templates*

The key component of a scaling approach using queueing systems as an analytical modelling technique is the capacity planning. This planning involves two steps: constructing a queueing network of a multi-tier application (step 2 in Figure 9) and solving the analytical model to generate a scaling plan (step 3 in Figure 9).

Figure 12 illustrates how the information maintained in a server template of Elastic-TOSCA maps to the corresponding part of a queueing network (dashed lines). The information maintained in the "Topology template" and "Relationship type" sections describe the topology of the queueing network and the linking relationship of different queueing systems. For each queueing system, $A/S/n$, the information in "Monitoring Information" section specifies the arrival process A, the information in the "Node types" section decides distribution of service time S and server number n.

After a queueing network representing the multi-tier application is constructed, the approach applies queueing theory to perform capacity planning using information in the "Monitoring Information" and "SLA&Constraints" sections. This planning estimates the number of servers to be deployed at each tier of the application and generates a scaling plan. The plan is then added to the "Plans" section to guide the scaling of the application (solid lines).

Take the e-commerce service in Figure 1 for example, using information in the "Topology templates" and "Relationship type" sections, the approach first constructs a queueing network of four queueing systems to describe the four tiers of servers in the service, and decides the linking sequence of these four queueing systems. The "Monitoring Information" and "Node type" sections then decide the three components of each queueing system. For example, the queueing system $A/S/n$ of Tomcat servers has arrival requests A with arrival rate 150 requests/second, each Tomcat server has service rate 70 requests/second, and the number of Tomcat servers is 1. Using the constructed queueing network, capacity planning is conducted according to the detected response time (3.5 seconds) in the "Monitoring Information" section and the required response time (2.0 seconds) in the "SLA&Constraints" section. The detected response time is larger than the required one, so a scaling up plan is generated: the tier of Tomcat should be added two servers and the tier of the Apache should be added one server.

V. IMPLEMENTATION AND EVALUATION

In this section, we first introduce iSSe and describe how it has been extended to interact with Elastic-TOSCA and the analytical models. We then describe the experiments conducted to illustrate both the effectiveness of the queueing system based scaling approach and the interaction of iSSe with Elastic-TOSCA.

A. *Extension of iSSe to Support Elastic-TOSCA*

We extended iSSe (see [10, 15] for detail), an intelligent scaling engine, to support the Elastic-TOSCA framework. As shown in Figure 13, iSSe acts as middleware between cloud IaaS providers and application owners. It provides a *Application owner portal* to assist application owners to configure their services, allow them to select servers from the iSSe Repository of Servers, define VM configurations, and design their topology. This portal also allows them to specify the required QoS and constraints. To enable interaction with Elastic-TOSCA, the information is stored in the "Topology template", "Node types", "Relationship types" sections in Elastic-TOSCA server templates.

The iSSe *Monitoring service* monitors each running application using two types of monitors. The first is the entry monitor, which examines the incoming requests over a finite interval (e.g., 60 seconds) and records information such as the requests' arrival rate and average response time. This information is used to decide whether a scaling up/down is needed. The second is the server monitor installed on each server to monitor its resource usage (e.g., CPU utilisation), and to analyse tier-specific values, such as response time. The collected information is then used to update the "Monitoring information" section in Elastic-TOSCA server templates.

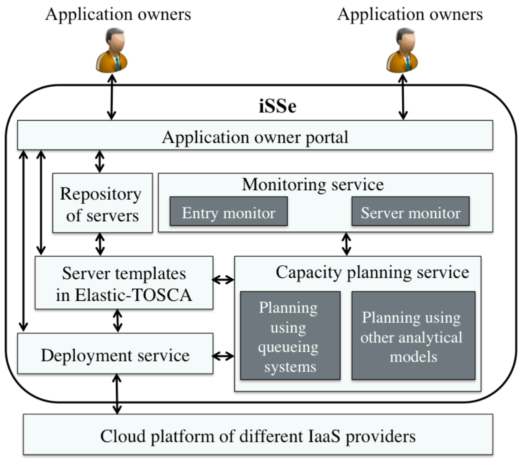

Application owners
Application owners
iSSe
Application owner portal
Repository of servers
Monitoring service
Entry monitor
Server monitor
Server templates in Elastic-TOSCA
Capacity planning service
Planning using queueing systems
Planning using other analytical models
Deployment service
Cloud platform of different IaaS providers

Figure 13. iSSe for supporting dynamic scaling using Elastic-TOSCA.

Note that the iSSe monitors are generic and independent of the IaaS providers. Existing providers usually provide users with standalone VM images. iSSe packages the VM images with pre-developed monitors as server templates that can be deployed in any cloud platform. The iSSe monitors hence only depend on the underlying operating systems, e.g., Linux Ubuntu and Centos, supported by iSSe. When a scaling is triggered, the iSSe *Capacity planning service* applies a scaling approach to generate a scaling up/down plan. When using Elastic-TOSCA, the information is maintained in the server templates. More concretely, the approach estimates the type and number of servers to be scaled, and then updates the "Plans" section by adding the generated scaling plan. Using the generated scaling plan, the iSSe *Deployment service* implements the required actions by calling interfaces of the underlying cloud platforms.

B. Evaluation of the Scaling Approach

Our evaluation is designed to illustrate the feasibility and effectiveness of Elastic-TOSCA in enabling dynamic scaling using a queueing system. The scaling approach itself is described in detail in [10], where it had been applied in an iSSe version not based on Elastic-TOSCA.

Our experiments are conducted in a data centre running IC-Cloud platform [16]. The configuration used has four physical machines (PMs), each with eight CPUs and 32 GB memory. The version of each processor is Quad-Core AMD Opteron(tm) Processor 2380, with 2.5GHz clock frequency and 512 KB cache size. All four PMs share a 4.1 Tb centralised storage and are connected through a switched gigabit Ethernet LAN with speed 1000mbs.

The e-commerce service in Figure 1 was implemented and its scaling up and down was tested. For convenience, each server of the service is installed on a single dedicated VM running Linux Centos 5.4. In deployment, different servers have different VM configuration details, as listed in Table I. Two versions of the MySQL database (Master and Slave) are implemented to support a data replication model. A MySQL Master is initially deployed and, when the tier of MySQL is scaled up, extra MySQL Slaves are added and configured with replication from the MySQL Master. Given a fixed VM configuration, the deployment of the Tomcat and Apache servers can be completed in a constant time. In the evaluation, the database has a fixed amount of data to be replicated, i.e., the data replication time of MySQL slave is fixed. Thus, the deployment time of MySQL databases is also a constant time.

TABLE I. FIVE TYPES OF SERVERS' VM CONFIGURATIONS

Service name	CPU	RAM (GB)	Software version
HAProxy	2	2	haproxy-1.4.8
Apache	2	2	Apache 2.2.20
Tomcat	1	1	Tomcat 7.0.22
MySQL Master	4	4	MySQL 5.5
MySQL Slave	1	1	MySQL 5.5

We used a client emulator to simulate a number of concurrent end users. Each end user continuously generates a sequence of requests to stress the server-side application. We divide the test into nine periods, where each period lasts 600 seconds. The first five periods of simulations stepwise increase in the number of end users so as to initiate scaling up. The remaining four periods gradually decrease this number to trigger scaling down. More concretely, the number of simulated concurrent users in the nine periods are: 200, 400, 600, 900, 1200, 900, 600, 400, and 200, respectively. This variance of end user numbers denotes the changing workload volume. The first testing period starts at time = 0 second. During the whole testing period, the application is monitored once every 60 seconds and Figure 14(a) displays the observed arrival rates of incoming requests. These observed arrival rates can be used to derive the mean and variance values of the request's interarrival time used in the queueing system.

Figure 14(b) lists the numbers of servers at each tier during scaling. Note that the numbers of HAProxy servers and MySQL Master database do not change. For the first period (the number of concurrent users is 200), the e-commerce service is initially deployed with one HAProxy, Apache, MySQL Master server and two Tomcat servers. When the concurrent users increase to 400 at time = 600 seconds and saturate the Apache and Tomcat tiers, dynamic scaling is triggered and one Apache and two Tomcats servers are added. When the number of concurrent users is increased at time=1200, 1800 and 2400 seconds, the cycle repeats. In contrast, when this number decreases at time =3000, 3600, 4200 and 4800 seconds, the service is scaled down by removing idle servers.

Note that, once scaling up or down is triggered, the construction of the application's queueing network model and executing the capacity planning are completed within few seconds to generate a scaling plan. Using the plan, servers are added or removed in parallel using the iSSe Deployment service. In the IC-Cloud platform [16], the deployment actions are completed within 1 or 2 minutes.

In the evaluation, we checked the monitoring information (request arrival rate and response time) every 60 seconds. We can observe in Figure 14 that there are 10 observation values for response time in each test period of 600 seconds. Typically, in each scaling up the first and second observed response time values violate the required constraint because scaling up is not yet completed. In other words, response recovery can be detected only after 1 to 2 minutes.

Figure 14(c) demonstrates the fluctuation of the end-to-end response time observed in the nine testing periods. In the first five periods, the response time is violated whenever the number of concurrent users is increased. For instance, when the number of users is increased to 400 at time = 600 seconds it saturates the Apache and Tomcat servers. Scaling up is then triggered and two Tomcat and one Apache servers are added. In contrast, in the last four periods, the scaling approach scales down the service while meeting the required response time.

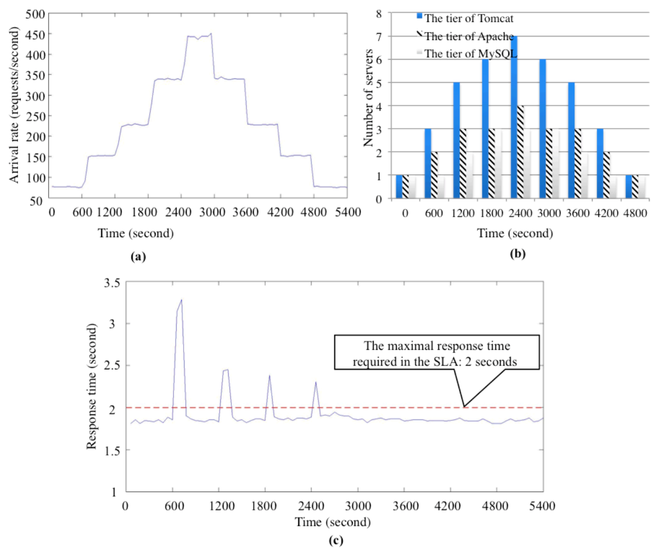


Figure 14. Evaluation of the scaling approach: (a) requests' arrival rate, (b) number of servers in each tier, (c) the end-to-end response time.

Result. The Elastic-TOSCA framework is able to support the scaling approach based on queueing systems for dynamically scaling up and down cloud applications to meet their QoS requirements.

VI. Conclusion and Future Work

In this paper, we presented extensions to the TOSCA framework to enable platform-independent specification of dynamic scaling for cloud applications. The extensions covered three sections, corresponding to three types of information used in guiding dynamic scaling. The Elastic-TOSCA framework is generic and supports a wide class of analytical mode-based scaling algorithms. We illustrated the effectiveness of Elastic-TOSCA framework by using a scaling approach based on a queueing system model. We also described how the framework can be supported easily in a scaling engine and conducted experiments to demonstrate the practicality of the framework using an example e-commerce service.

Our direct future work is to evaluate supporting Elastic-TOSCA on different cloud platforms and also to evaluate using other scaling techniques and approaches such as lightweight scaling at the VM level itself (CPUs, memory, I/O, etc) [17]. We will also test our approach using more complex scenarios such as considering the amount of data to be replicated in the MySQL slave databases in the e-commerce application, as well as by using other multi-tier applications.

References

[1] Amazon Web Services (Amazon WS): http://aws.amazon.com/ec2/ [retrieved: 03, 2013]

[2] GoGrid: http://www.gogrid.com/ [retrieved: 03, 2013].

[3] Binz, T., Breiter, G., Leyman, F. and Spatzier, T., "Portable Cloud Services Using TOSCA," Internet Computing, IEEE, vol. 16, pp. 80-85, 2012.

[4] Topology and Orchestration Specification for Cloud Applications (TOSCA), OASIS specification: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca [retrieved: 03, 2013].

[5] R. B. Cooper, Introduction to queueing theory, second edition ed. New York: North-Holland, 1981.

[6] K. Xiong and H. Perros, "Service performance and analysis in Cloud computing," in 2009 Congress on Services - I, Los Angeles, CA 2009, pp. 693-700.

[7] D. A. Bacigalupo, et al., "Managing dynamic enterprise and urgent workloads on clouds using layered queuing and historical performance models," Simulation Modelling Practice and Theory, vol. 19, pp. 1479-1495, 2011.

[8] Bi, jin, Zhu, Zhiliang, Tian, Ruixiong and Wang, Qingbo, "Dynamic Provisioning Modeling for Virtualized Multi-tier Applications in Cloud Data Center," in 2010 IEEE 3rd International Conference on Cloud Computing (CLOUD'10), Miami, Florida, 2010, pp. 370-377.

[9] Hu, Y., Wong, J., Iszlai, G. and Litoiu, M., "Resource provisioning for cloud computing," in Proceedings of the 2009 Conference of the Center for Advanced Studies on Collaborative Research (CASCON '09), 2009, pp. 101-111.

[10] R. Han, M. Ghanem., L. Guo, Y. Guo, and M. Osmond, "Enabling cost-aware and adaptive elasticity of multi-tier cloud applications," Future Generation Computer Systems, 2012, pp. 1-17, doi:10.1016/j.future.2012.05.018.

[11] R. Pal and P. Hui, "On the Economics of Cloud Markets," , Technical Report, University of Southern Californi, Los Angeles, 2011, pp. 1-7.

[12] Ghosh, R., Trivedi, K.S., Naik, V.K. and Kim, D.S., "End-to-end performability analysis for infrastructure-as-a-service cloud: An interacting stochastic models approach," in 2010 IEEE 16th Pacific Rim International Symposium on Dependable Computing, Tokyo, Japan, 2010, pp. 125-132.

[13] Ghosh, R., Longo, F., Naik, V.K. and Trivedi, K.S., "Quantifying resiliency of IaaS cloud," in 2010 29th IEEE Symposium on Reliable Distributed Systems, New Delhi, Punjab India, 2010, pp. 343-347.

[14] J. Z. Li, "Fast Optimization for Scalable Application Deployments in Large Service Centers," Doctor of Philosophy, Department of Systems and Computer Engineerin, Carleton University, Ottawa, Ontario, 2011.

[15] R. Han, L. Guo, Y. Guo, and S. He, "A Deployment Platform for Dynamically Scaling Applications in The Cloud," in the 3rd IEEE International Conference on Cloud Computing Technology and Science (CloudCom'11), Athens, Greece, 2011, pp. 506-510.

[16] L. Guo, Y. Guo, X. Tian, "IC Cloud: A Design Space for Composable Cloud Computing," in 2010 IEEE 3rd International Conference on Cloud Computing (CLOUD'10), Miami, Florida, 2010, pp. 394-401.

[17] R. Han, L. Guo, M. Ghanem., and Y. Guo, “Lightweight Resource Scaling for Cloud Applications,”, in 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2012), Ottawa, Canada, 2012, pp. 644-651.