Towards the Formalization of Properties of Cloud-Based Elastic Systems

Marcello M. Bersani
Politecnico di Milano
Milano, Italy
bersani@elet.polimi.it

Domenico Bianculli
University of Luxembourg
Luxembourg, Luxembourg
domenico.bianculli@uni.lu

Schahram Dustdar
TU Wien
Vienna, Austria
dustdar@infosys.tuwien.ac.at

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

Carlo Ghezzi
Politecnico di Milano
Milano, Italy
carlo.ghezzi@polimi.it

Srđan Krstić
Politecnico di Milano
Milano, Italy
srdan.krstic@polimi.it

ABSTRACT

Cloud-based elastic systems run on a cloud infrastructure and have the capability of dynamically adjusting the allocation of their resources in response to changes in the workload, in a way that balances the trade-off between the desired quality-of-service and the operational costs. The actual elastic behavior of these systems is determined by a combination of factors, including the input workload, the logic of the elastic controller determining the type of resource adjustment, and the underlying technological platform implementing the cloud infrastructure. All these factors have to be taken into account to express the desired elastic behavior of a system, as well as to verify whether the system manifests or not such a behavior.

In this paper, we take a first step into these directions, by proposing a formalization, based on the CLTL$^t(\mathcal{D})$ temporal logic, of several concepts and properties related to the behavior of cloud-based elastic systems. We also report on our preliminary evaluation of the feasibility to check the (formalized) properties on execution traces using an automated verification tool.

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification—*Formal Methods*

General Terms

Theory

Keywords

Cloud computing, elastic systems, temporal logic

PESOS'14, May 31, 2014, Hyderabad, India
Copyright 2014 ACM 978-1-4503-2841-8/14/05...$15.00
http://dx.doi.org/10.1145/2593793.2593798

1. INTRODUCTION

Cloud computing has become a practical solution to manage and leverage IT resources and services. Cloud platforms offer several benefits, among which the ability to access resources or service applications offered as (remote) services, available on-demand and on-the-fly, and billed according to a pay-per-use model.

Cloud providers offer resources and services at three different layers: at the *Software-as-a-Service (SaaS)* layer, users can remotely access full-fledged software applications; at the *Platform-as-a Service (PaaS)* layer, one finds a development platform, a deployment and a run-time execution environment, which is used to run user-provided code in sandboxes hosted on cloud-based premises; at the *Infrastructure-as-a-Service (IaaS)* the user can access computing resources such as virtual machines, block storage, firewalls, load balancers, or networking I/O.

In this paper, we focus on the IaaS layer, and assume, without loss of generality, that resources offered at this level are virtual machines. In particular, we consider *cloud-based elastic systems*: elasticity [7] of computing systems is defined [11] by the (US) National Institute of Standards and Technology (NIST) as:

> Capabilities can be rapidly and elastically provisioned, in some cases automatically, to quickly scale out, and rapidly released to quickly scale in. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be purchased in any quantity at any time.

In the case of systems based on a cloud infrastructure, this definition can be interpreted, borrowing some terms from physics, as the capability of the system "to stretch" by dynamically acquiring new computing resources (e.g., by starting new virtual machines), and "to contract" by releasing resources (e.g., by terminating virtual machines). Scaling out ("stretching") and scaling in ("contracting") actions (also called *adaptation* actions) are determined by an elastic controller, in response to changes in the workload.

Application providers exploit resource elasticity at run time to balance the trade-offs between the quality of service (QoS), the input workload, and the operational costs. The main goal is, when confronted with fluctuating workloads, to maintain the QoS at an adequate level while minimizing the costs. In particular, when the input workload escalates over the current system capacity, the acquisition of new resources prevents *under*-provisioning, and allows the system to maintain an adequate QoS, even though operational costs increase. On the other hand, when the input workload decreases below the current system capacity, releasing some resources prevents *over*-provisioning, and contributes to reducing the costs while still providing an adequate QoS.

The behavior (in terms of dynamically scaling up and down resource allocation) of a cloud-based elastic system depends on the combination of many factors, such as the input workload, the logic that is embodied in the elastic controller to trigger adaptations and the underlying technological platform implementing the cloud infrastructure.

The complexity of these dependencies and the hard-to-determine effects on the behavior of the system as perceived by users, makes the design of cloud-based elastic systems very challenging. From the point of view of specification and verification the three main open issues are: 1) how to specify the desired elastic behavior of these systems; 2) how to check whether they manifest or not such an elastic behavior; 3) how to identify when and how they depart from their intended behavior.

In this paper, we take a first step in these directions, and propose a formalization of the behavior of cloud-based elastic systems, by characterizing the properties related to elasticity, resource management, and quality of service. We formally characterize these properties using a temporal logic called CLTL$^t(\mathcal{D})$, which stands for Timed Constraint LTL. The use of a temporal logic paves the way for using established verification tools to check whether the proposed properties, which correspond to specific facets of an elastic behavior, hold or not during the execution of cloud-based elastic systems. In regards to this, in the paper we also report on a preliminary evaluation we performed to assess the feasibility of checking the proposed properties expressed in CLTL$^t(\mathcal{D})$ on execution traces of a realistic application using ZOT [13], a verification toolset based on SAT- and SMT-solvers.

The rest of the paper is organized as follows. Section 2 provides an overview of cloud-based elastic systems, describing how they operate. Section 3 introduces CLTL$^t(\mathcal{D})$, the temporal logic used later for the formalization. Section 4 formally defines some general aspects of resources used in cloud-based systems. Section 5 illustrates the formalization of the properties that we have considered. Section 6 reports on a preliminary evaluation of checking some properties on realistic execution traces. Section 7 surveys the related work and Section 8 concludes the paper by describing future lines of research.

2. CLOUD-BASED ELASTIC SYSTEMS

Two hallmarks of cloud computing are the ability to dynamically manage the allocation of resources in the system and the pay-per-use billing model. In particular, these traits characterize *cloud-based elastic systems*, which are systems that can dynamically adjust their resources allocation to maintain a predefined/suitable level of QoS, in spite of fluctuating input workloads, while minimizing running costs. The key aspect of cloud-based elastic systems is their ability to *adapt* at run time, in response to a change in the operating conditions (e.g., a spike in the number of input requests). In this context, adaptation means trying to manage the allocation of resources so that they match the capacity required to properly sustain the input workload. In other words, cloud-based elastic systems aim to prevent both over-provisioning (allocating more resources than required) and under-provisioning (allocating fewer resource than required).

The behavior of an elastic system can be intuitively described as follows. Consider the case in which there is an increase in the load of a system, which might lead to the saturation of system resources, causing a degradation of the QoS perceived by end-users. To avoid the saturation, an elastic system *stretches*, i.e., its capacity is scaled up by allocating additional resources (acquired from a cloud infrastructure); the load can then be spread over a bigger set of resources. Conversely, when the system load decreases, some resources might become under-utilized, hence unnecessarily expensive. To reduce costs, an elastic system *contracts*, i.e., its capacity is scaled down by deallocating a portion of the allocated resources, which are then released back to the cloud infrastructure.

Cloud-based elastic systems usually implement the closed-loop architecture shown in Figure 1, where an *elastic controller* monitors the actual system (i.e., the *controlled system*) and determines its adaptation. End-users send their requests, which constitute the input workload of the elastic system, through its public interface. Notice that the workload may fluctuate because of seasonality in the users' demand or some unexpected events such as a flash-crowd (i.e., the appearance of a web site on a popular blog or news column, determining an exponential spike of the requests to the server).

The *controlled system* responds to end-user requests by implementing the business logic of the application. It is deployed onto a cloud infrastructure provided by a dedicated IaaS provider, and constituted by a set of cooperating virtual machines. The controlled system implements also the logic to change the allocation of resources (i.e., virtual machines) and adjust the capacity of the system; these are essential functionalities to enable an elastic behavior. The controlled system is also characterized by two attributes that constrain, respectively, the minimum and the maximum number of allocated resources. The former corresponds to the minimal amount of resources that must be always allocated to guarantee the provision of the application functionalities to end-users. As for the latter, it sets an upper bound for the maximum amount of allocated resources, beyond which scaling the system is not cost-effective anymore.

The elastic controller periodically monitors the operating parameters of the controlled system (e.g., the *system load*) and determines the control actions to be executed to perform adaptation. The controller implements the logic that tries to fulfill the high-level goals (e.g., minimizing running costs while delivering a certain level of QoS) specified by the service provider that operates the elastic system. The control actions that the controller can issue are *scale-up* and *scale-down*, which correspond to instantiating and terminating virtual machines, respectively. These actions are sent to the controlled system through its cloud interface, which plays the role of the controller actuator. Notice that executing these actions might take a non-negligible time, which is called actuation delay. The cloud interface propagates the control actions issued by the controller to

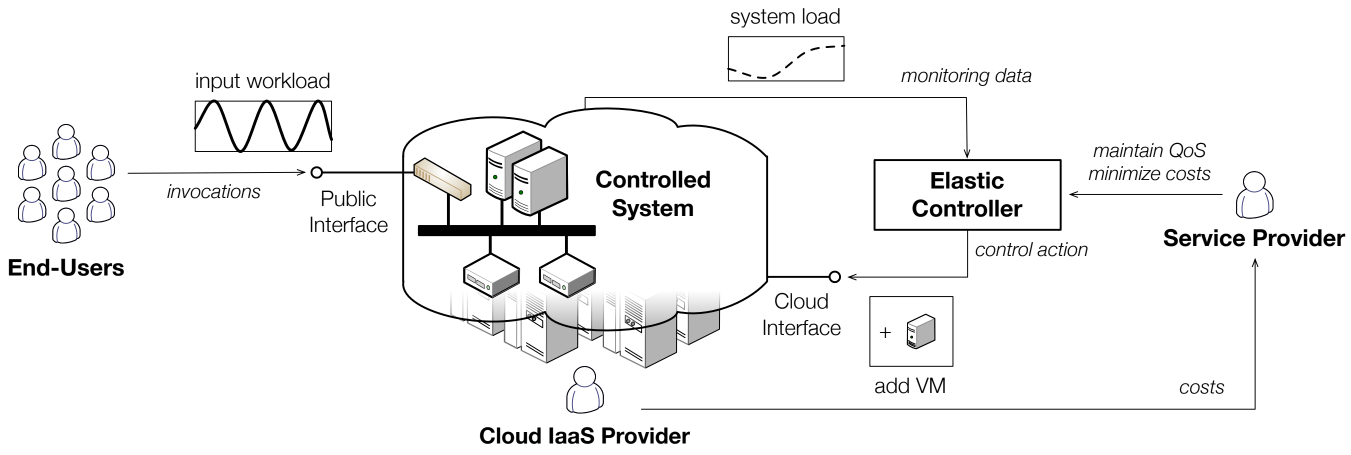

Figure 1: High level architecture/view of cloud-based elastic systems.

the cloud IaaS provider, which performs the physical allocation/deallocation of virtual machines. The cloud IaaS provider tracks the total resource usage accumulated by each service provider, who is then billed for the cost of running the system.

3. CLTL$^t(\mathcal{D})$ IN A NUTSHELL

In this section we provide an overview of Timed Constraint LTL (CLTL$^t(\mathcal{D})$), the temporal logic we have chosen to formalize the properties of cloud-based elastic systems. CLTL$^t(\mathcal{D})$ is rooted in two other temporal logics, Linear Temporal Logic (LTL) and Constraint LTL (CLTL($\mathcal{D}$)).

LTL [10] is one of the most popular descriptive language for defining temporal behaviors that are represented as sequences of observations. The time model adopted in this logic is a totally ordered set (e.g., $(\mathbb{N}, <)$) whose elements are the positions where the behavior is observed. LTL allows the expression of positional orders of events both towards the past and the future. For example, a property like "if a query Q is received, then a response R will be returned after two *positions* from the one in which Q is received" is satisfied by the following sequence of events: $\emptyset$ $\{Q\}$ $\emptyset$ $\{R\}$ $\emptyset$ $\emptyset$ $\{Q\}$ $\emptyset$ $\{R\}\dots$, because the occurrences of event Q in position 1 and 6 are followed, respectively, by the occurrences of event R in position 3 and 8; we denote with $\emptyset$ the positions in which no event occurs. Leveraging the symmetry between past and future, one can verify that the same sequence satisfies the property "every R is preceded by a Q two *positions* before".

CLTL($\mathcal{D}$) [4] is an extension of LTL that includes arithmetical atomic formulae built over a constraint system $\mathcal{D}$; this extension enables the use of variables in the formulae. Intuitively, using this extension one can define the behavior of *variables* across positions through *arithmetical constraints* over $\mathcal{D}$. For example, we can express a property like "whenever an event S occurs, in the next position, variable z must be incremented by 1 with respect to the value at the current position".

To define CLTL$^t(\mathcal{D})$, we couple each position in the time model with a timestamp and we introduce timing constraints to the CLTL($\mathcal{D}$) temporal operators. Thanks to the timestamp information, we can account for the absolute time at which events are observed, and define the behavior of variables *over the time* through arithmetical constraints over $\mathcal{D}$. For instance, consider the following sequence: $(\{Q\},\{2\},3)$ $(\emptyset,\{2\},4)$ $(\{R\},\{1\},5)$ $(\{Q\},\{2\},15)$ $(\{R\},\{0\},20)$. In each position we have a triple: the first element is a set of events occurring in that position (e.g., Q or R); the second is a set of values of variables (e.g., modeling the number of pending jobs inside the system); the third element is the timestamp. Using the timestamps, all positions in the sequence correspond to time instants where the behavior of the system changes. For example, at the first position Q occurs, the elapsed time is 3 and the number of pending jobs is 2. Position 3 captures the fact that at time 5 the system replies (denoted with R), completing only one of the two pending jobs, thus one job remains. Position 4 denotes that at time 15 another query (this time equipped with one job) occurs; the number of pending jobs is incremented accordingly. The last position indicates that at time 20 the system generates another reply and, by that time, all the pending jobs are done (the corresponding variable is equal to 0).

Let $\mathcal{D}$ be structure $(\mathbb{Z}, =, (<_d)_{d\in\mathbb{Z}})$, and let V be a set of variables over $\mathcal{D}$. The binary relation $<_d$ is defined as $x <_d y \Leftrightarrow x < y + d$. With these settings, we can for example abbreviate the conjunction $y <_{1-d} x \wedge x <_{d+1} y$ with the notation $x = y + d$. The logic introduced so far has been proved to be decidable in [9].

In the rest of this section, we formally define CLTL$^t(\mathcal{D})$. Atomic formulae in CLTL$^t(\mathcal{D})$ can be *propositions* or *constraints* over $\mathcal{D}$. The syntax of the terms used in the constraints, called *arithmetic temporal terms* (hereafter simply called terms) is defined as:

$$\alpha := c \mid x \mid \mathsf{Y}(\alpha) \mid \mathsf{X}(\alpha)$$

where $c \in \mathbb{Z}$ is a constant, $x \in V$ is a variable, Y is the *arithmetical previous* temporal operator, and X is the *arithmetical next* temporal operator. The temporal operators are applied to terms, and they refer to the value of that term in the previous (Y) and in the next (X) position in the sequence, i.e., the corresponding discrete position. The *depth* of term α (denoted as $|\alpha|$) is the total amount of temporal shift needed in evaluating α. For example $|\mathsf{XX}(x)| = 2$, $|\mathsf{YYY}(x)| = -3$ and $|\mathsf{YX}(x)| = |\mathsf{XY}(x)| = 0$.

A well-formed CLTL$^t(\mathcal{D})$ formula ϕ is defined according to this syntax:

$$\phi ::= p \mid \alpha \sim \alpha \mid \neg\phi \mid \phi \wedge \phi \mid \phi\, \mathbf{U}_I\, \phi \mid \phi\, \mathbf{S}_I\, \phi$$

where p is an atomic proposition from a finite set Π; α is a *term*; the relation $\sim$ belongs to $\{=, (<_d)_{d\in\mathbb{Z}}\}$; I identifies a non-empty interval over $\mathbb{N}$;[1] $\mathbf{U}_I$ ("*Until*") and $\mathbf{S}_I$ ("*Since*") are the modalities of LTL.

Additional modalities can be defined using the standard conventions. Let $\perp$ be an abbreviation for $p \wedge \neg p$ representing "false". "*Yesterday*" can be defined as $\mathbf{Y}_I\phi \leftrightarrow \perp \mathbf{S}_I\phi$, "*Next*" as $\mathbf{X}_I\phi \leftrightarrow \perp \mathbf{U}_I\phi$. Similarly, one can also define other modalities such as $\mathbf{G}_I$ ("*Globally*"), $\mathbf{F}_I$ ("*Eventually in the future*"), $\mathbf{P}_I$ (*"Eventually in the past"*).

An example of a well-formed CLTL$^t(\mathcal{D})$ formula is $\mathbf{G}(\phi \rightarrow \mathsf{X}(z) = z+1)$. This formula states that whenever ϕ is true, the value of variable z in the next time instant is constrained to be increased by one with respect to the value at the current instant.

Hereafter, we use quantifiers ($\forall$ and $\exists$) and parameterized propositions over finite sets as a shorthand for representing a group of constraints. For example, given the set $A = \{1, 2, 3\}$ and the parameterized proposition $p(\cdot)$, the formula $\forall a \in A : p(a)$ is a shorthand for $p_1 \wedge p_2 \wedge p_3$, where $p_1, p_2, p_3 \in \Pi$. We omit the definition of the set when it is clear from the context.

The formal semantics of CLTL$^t(\mathcal{D})$ formulae can be defined as follows. Let $\tau = \tau_0\tau_1\ldots$ be a timed sequence, i.e., an infinite strictly monotonic sequence of values so that $\tau_i < \tau_{i+1}$, for all $i \geq 0$. Let $\pi = \pi_0\pi_1\ldots$ be an infinite sequence over $\wp(\Pi)$ which associates a subset of the set of propositions with each position. Let $\sigma = \sigma_0\sigma_1\ldots$ be an infinite sequence of evaluations $\sigma_i : V \rightarrow \mathbb{Z}$ defining the value of variables at each time position. We denote the value of x at position i with $\sigma_i(x)$ and the value $\sigma_{i+|\alpha|}(x_\alpha)$ with $\sigma_i(\alpha)$, where x_α is the variable in V occurring in term α, if any. Given a time instant $i \geq 0$ and a structure (π, σ, τ), we define the satisfaction relation $(\pi, \sigma, \tau), i \models \phi$ for CLTL$^t(\mathcal{D})$ formulae as shown in Figure 2. A formula $\phi \in$ CLTL$^t(\mathcal{D})$ is *satisfiable* if there exists a triple (π, σ, τ) such that $(\pi, \sigma, \tau), 0 \models \phi$; in this case, we say that (π, σ, τ) is a *model* of ϕ, with (π, τ) being the *timed propositional model* and σ being the *arithmetic model*. CLTL($\mathcal{D}$) over infinite words has been proven undecidable [4], however we use CLTL$^t(\mathcal{D})$ over finite words (i.e., for trace checking), thus we retain decidability.

$$\begin{aligned}
(\pi,\sigma,\tau), i \models p &\Leftrightarrow p \in \pi_i \text{ for } p \in \Pi\\
(\pi,\sigma,\tau), i \models \alpha_1 \sim \alpha_2 &\Leftrightarrow \sigma_i(\alpha_1) \sim \sigma_i(\alpha_2)\\
(\pi,\sigma,\tau), i \models \neg\phi &\Leftrightarrow (\pi,\sigma,\tau), i \not\models \phi\\
(\pi,\sigma,\tau), i \models \phi \wedge \psi &\Leftrightarrow (\pi,\sigma,\tau), i \models \phi \text{ and } (\pi,\sigma,\tau), i \models \psi\\
(\pi,\sigma,\tau), i \models \phi\, \mathbf{U}_I\, \psi &\Leftrightarrow \exists\, j > i : (\pi,\sigma,\tau), j \models \psi \text{ and}\\
&\qquad \forall n \in (i,j) : \begin{pmatrix} (\pi,\sigma,\tau), n \models \phi \text{ and}\\ \tau_j - \tau_i \in I \end{pmatrix}\\
(\pi,\sigma,\tau), i \models \phi\, \mathbf{S}_I\, \psi &\Leftrightarrow \exists\, 0 \leq j < i : (\pi,\sigma,\tau), j \models \psi \text{ and}\\
&\qquad \forall n \in (j,i) : \begin{pmatrix} (\pi,\sigma,\tau), n \models \phi \text{ and}\\ \tau_j - \tau_i \in I. \end{pmatrix}.
\end{aligned}$$

Figure 2: Semantics of CLTL$^t(\mathcal{D})$.

4. MODELING RESOURCES OF CLOUD-BASED SYSTEMS

At the basis of cloud-based service provisioning there is the possibility of accessing remote resources. As anticipated in Section 1, in this paper we consider elastic behaviors with respect to the management and adaptation of resources offered at the IaaS level, in particular virtual machines; hereafter, by slightly abusing the terminology, we will refer to resources and virtual machines interchangeably. In the rest of this section we introduce some useful notation and express in CLTL$^t(\mathcal{D})$ some general aspects that characterize the lifecycle of resources in a cloud-based system.[2] These aspects will then be *assumed* to hold across the formalization of the properties of cloud-based elastic systems in the next section.

We model the total resources in use by a system at a certain time instant by means of a non-negative integer variable $R \in V$. We use constants R_{min} and R_{max} to denote the minimum and maximum number of resources that the system can allocate. At any time, the amount of resources allocated to a system must be within these limits. We capture this constraint on resource allocation with the following formula:

$$\mathbf{G}(R_{min} \leq R \;\wedge\; R \leq R_{max}) \qquad (\mathcal{M}_{bound})$$

which states that number of resources R is bounded throughout the entire execution of the cloud-based system.

We use a non-negative integer variable $L \in V$ to denote the current load of the system, expressed in terms of required resources. We assume that each cloud-based system has always enough resources to support the current load (the *manageable load* assumption), as stated below:

$$\mathbf{G}(L \leq R_{max}) \qquad (\mathcal{M}_{load})$$

Virtual Machine Lifecycle

At the IaaS layer, users create virtual machines to host their applications; each virtual machine is uniquely identified by an ID. Let R_{max} be a positive integer parameter representing the maximum number of virtual machines that can be allocated by a system. The set of valid virtual machines IDs is then defined as $ID = \{0, \ldots, R_{max} - 1\}$.

To model the events characterizing the lifecycle of a virtual machine, we represent them as parameterized propositions. We use $M_{start}(\cdot)$ to denote a request to instantiate a new virtual machine; conversely, we use $M_{stop}(\cdot)$ to denote a shut-down request. After receiving the request to instantiate a new virtual machine, the cloud infrastructure starts the actual instantiation process by allocating the physical resources and booting the OS: the end of the OS boot is denoted with proposition $M_{boot}(\cdot)$. The actual event in which the user application is ready to process the input is denoted with $M_{ready}(\cdot)$. Similarly, in the case of a shut-down request, we denote the actual termination of the virtual machine (following the shut-down request) with the proposition $M_{end}(\cdot)$. The order of occurrence of these events has to match the one

[1]We omit the interval from the formulae when it is $[0, +\infty)$.
[2]We use the term "cloud-based system" instead of the one "cloud-based elastic system" used elsewhere in the paper, since the aspects described here for modeling the resources can be assumed to hold for every kind of cloud-based system, not necessarily with an elastic behavior.

defined by the lifecycle of a virtual machine: M_{start}–M_{boot}–M_{ready}–M_{stop}–M_{end}. We state this constraint by AND-ing the following formulae $\mathcal{M}_{sb}$, $\mathcal{M}_{br}$, $\mathcal{M}_{rs}$, $\mathcal{M}_{se}$:

$$\begin{aligned}\forall id : \mathbf{G}(M_{start}(id) \rightarrow &((\neg M_{start}(id) \wedge \neg M_{ready}(id) \\ &\wedge \neg M_{stop}(id) \wedge \neg M_{end}(id)) \\ &\mathbf{U}\, M_{boot}(id)))\end{aligned} \qquad (\mathcal{M}_{sb})$$

$$\begin{aligned}\forall id : \mathbf{G}(M_{boot}(id) \rightarrow &((\neg M_{boot}(id) \wedge \neg M_{start}(id) \\ &\wedge \neg M_{stop}(id) \wedge \neg M_{end}(id)) \\ &\mathbf{U}\, M_{ready}(id)))\end{aligned} \qquad (\mathcal{M}_{br})$$

$$\begin{aligned}\forall id : \mathbf{G}(M_{ready}(id) \rightarrow &((\neg M_{ready}(id) \wedge \neg M_{start}(id) \\ &\wedge \neg M_{boot}(id) \wedge \neg M_{end}(id)) \\ &\mathbf{U}\, M_{stop}(id)))\end{aligned} \qquad (\mathcal{M}_{rs})$$

$$\begin{aligned}\forall id : \mathbf{G}(M_{stop}(id) \rightarrow &((\neg M_{stop}(id) \wedge \neg M_{ready}(id) \\ &\wedge \neg M_{start}(id) \wedge \neg M_{boot}(id)) \\ &\mathbf{U}\, M_{end}(id)))\end{aligned} \qquad (\mathcal{M}_{se})$$

All the formulae above follow the same structure. For example, in the case of Formula $\mathcal{M}_{sb}$, we state that after a request to instantiate a certain virtual machine (M_{start}), all other requests for the same machine (events M_{start}, M_{ready}, M_{stop}, M_{end}) cannot occur until the OS boot ends (event M_{boot}).

In addition, we require that the lifecycle of a virtual machine starts with event M_{start}:

$$\forall id : \mathbf{G}(M_{end}(id) \rightarrow \mathbf{P}(M_{start}(id))) \qquad (\mathcal{M}_{start})$$

The formula states that event M_{end} is always preceded by event M_{start}, for any id.

Finally, we specify that the transition from event $M_{start}(\cdot)$ to event $M_{boot}(\cdot)$ might take some finite time, bounded by the parameter T_{cd} defined by each single provider. This requirement is expressed as:

$$\forall id : \mathbf{G}(M_{start}(id) \rightarrow \mathbf{F}_{(0,T_{cd})}(M_{boot}(id)))^{3} \qquad (\mathcal{M}_{bad})$$

The formula states that after receiving a request to allocate a new virtual machine, the boot process has to complete within T_{cd} time units.

5. CONCEPTS AND PROPERTIES OF CLOUD-BASED ELASTIC SYSTEMS

In this section we present concepts and properties that can be used to characterize relevant behaviors of cloud-based elastic systems. The concepts and the properties have been selected and derived based on our research experience in the field, especially matured within EU-funded projects like RESERVOIR [12] and CELAR [3]. The presentation is divided in three groups: elasticity, resource management, and quality of service.

As previously remarked, for the proposed formalization of properties, we assume that the concepts described in the previous section must always hold, i.e., if $\mathcal{C}_{system}$ is the conjunction of all formulae described in Section 4, we consider execution traces that satisfy $\mathcal{C}_{system}$.

5.1 Elasticity

As we have seen in Section 2, elastic systems are supposed to dynamically adapt in reaction to fluctuations in the input workload, by changing their computing capacity. In the case of cloud-based elastic systems, the adaptation is performed either by increasing the computing capacity with the allocation of additional resources, or by decreasing the capacity by releasing a portion of those resources.

Recall that we model the resources currently in use by a system with a non-negative integer variable R. Since elastic systems usually start with a minimal allocation of resources, at the beginning we set the value of variable R equal to R_{min}:

$$R = R_{min} \qquad (\mathcal{M}_{init})$$

A cloud-based elastic system must change its resources allocation according to the decisions made by the elastic controller. These decisions result in the requests M_{start} and M_{stop}, to allocate or deallocate resources, which ultimately determine the amount of resources that are actually in use by the system, which we model with the variable R. The value of R has to change when either an M_{start} or M_{stop} event occurs. We set an arithmetic constraint on the value of R with the following formulae:

$$\forall id : \mathbf{G}(M_{start}(id) \rightarrow R = \mathsf{Y}(R) + 1) \qquad (\mathcal{M}_{i})$$

$$\forall id : \mathbf{G}(M_{stop}(id) \rightarrow R = \mathsf{Y}(R) - 1) \qquad (\mathcal{M}_{d})$$

The first formula states that after receiving the request to instantiate a new virtual machine the number of resources in use by the system must be increased; conversely, the second formula states that when there is a request to shut down a virtual machine, the number of resources must be decreased.

We also require that changes to the allocation of resource should happen only as a consequence of an M_{start} or M_{stop} request, triggered by the elastic controller.[4] This requirement is expressed as a constraint on changes to the value of R with the following formulae:

$$\exists id : \mathbf{G}(R = \mathsf{Y}(R) + 1 \rightarrow M_{start}(id)) \qquad (\mathcal{M}_{ix})$$

$$\exists id : \mathbf{G}(R = \mathsf{Y}(R) - 1 \rightarrow M_{stop}(id)) \qquad (\mathcal{M}_{dx})$$

We explicitly require that the value of R must not change if neither M_{start} nor M_{stop} occurs with the formula:

$$\mathbf{G}((\forall id : \neg M_{start}(id) \wedge \neg M_{stop}(id)) \leftrightarrow R = \mathsf{Y}(R)) \qquad (\mathcal{M}_{eq})$$

After formalizing resource change over time, we introduce the concepts of *eagerness* and *sensitivity*, which capture dynamic aspects of an elastic behavior, such as the speed of adaptation and the minimum variation in the load that triggers an adaptation.

Eagerness informally refers to the speed of the reaction of a system upon a change of the load. It captures the fact that elastic systems must adapt to changes in the workload in a timely manner. We introduce parameter T_e to represent the maximum amount of time within which an elastic system must react to change in the load.

Sensitivity informally refers to the minimum change in the load that should trigger adaptation. It captures the fact

[3] A closed interval $[a, b]$ over $\mathbb{N}$ can be expressed as an open one of the form $(a-1, b+1)$, $a \geq 1$.

[4] We assume that the elastic controller is the only component that can issue the M_{start} and M_{stop} requests.

that different elastic systems may react to different load intensities and variations. Sensitivity prevents the system from adapting to small, transient changes in the load, possibly creating unnecessary costs. We model the sensitivity of a system with the parameter Δ. This parameter defines a range over the currently measured load: if the load stays within this range, then adaptations are not necessary and will not be triggered. The parameter Δ can assume values from the $[0, R_{max})$ interval. The case $\Delta = 0$ identifies an elastic system that reacts to any change in the load. Under the "manageable load" assumption (see Formula $\mathcal{M}_{load}$), the case $\Delta = R_{max}$ would identify a system that is totally insensitive to the load, i.e., a system that does not adapt. In our understanding this is not an elastic system, and we do not allow this behavior.

We introduce an auxiliary variable, $L_a \in V$, to model eagerness and sensitivity. This variable accumulates the change in the value of the load L. The behavior of L_a is constrained as shown below:

$$L_a = 0 \qquad (\mathcal{P}_{rt1})$$

$$\mathbf{G}((-\Delta \leq L_a \leq \Delta) \rightarrow \mathsf{X}(L_a) = L_a + \mathsf{X}(L) - L \qquad (\mathcal{P}_{rt2})$$

The first formula initializes the value of L_a to zero. The second formula constraints the value of L_a to change only if its value stays within the threshold defined by the parameter Δ; the value of L_a is incremented according to the difference of the values of the system load L in two consecutive time positions ($\mathsf{X}(L) - L$).

We can now characterize the behavior of a system when scaling up and down occur in terms of variables R and L_a:

$$\mathbf{G}((L_a > \Delta) \rightarrow (\mathsf{X}(L_a) = \mathsf{X}(L) - L \wedge \mathbf{F}_{(0,T_e]}(\mathsf{X}(R) > R))) \qquad (\mathcal{P}_{rt3})$$

$$\mathbf{G}((L_a < -\Delta) \rightarrow (\mathsf{X}(L_a) = \mathsf{X}(L) - L \wedge \mathbf{F}_{(0,T_e]}(\mathsf{X}(R) < R))) \qquad (\mathcal{P}_{rt4})$$

The two formulae express the possible changes in the number of allocated resources: either an increase (denoted with the arithmetical constraint $\mathsf{X}(R) > R$ in $\mathcal{P}_{rt3}$) or a decrease (denoted with $\mathsf{X}(R) < R$ in $\mathcal{P}_{rt4}$). In both cases, the adaptation is triggered when the value of L_a (the accumulated change of the load) exceeds the threshold defined by parameter Δ. Moreover, the number of resources R is constrained to change within the temporal bound T_e. Furthermore, the formulae constrain also the value of L_a in the next instant, by "resetting" the value accumulated so far.

Plasticity. A distinctive characteristic of elastic systems is their ability to release resources when the load decreases. In particular, when the load drops to zero an elastic system must be able to deallocate all its resources within a reasonable time and return to the minimal configuration of resource allocation. We call systems that do not show this behavior *plastic*; a plastic system is a system that cannot return to its minimal configuration after increasing the number of resources. Ideally, a truly elastic system should never show a plastic behavior.

We introduce parameters T_{p_1} and T_{p_2}. Parameter T_{p_1} indicates for how long the system needs to experience no load, before deallocating all the resources; it is useful to avoid reacting to transient and short-term changes in the load. The other parameter T_{p_2} represents the maximum time the system has to react if a complete deallocation of resources is needed. The following formula characterizes a system that is *not* plastic:

$$\mathbf{G}(\mathbf{G}_{(0,T_{p_1}]}(L = 0) \rightarrow \mathbf{F}_{(0,T_{p_2}]}R = R_{min}) \qquad (\mathcal{P}_{pl})$$

It states that, for all time positions, if a load is equal to zero in a time range bounded by T_{p_1}, then the number of resources will return to its minimal configuration within T_{p_2}. The violation of Formula ($\mathcal{P}_{pl}$) is a sufficient condition for a system to be plastic.

5.2 Resource Management

There is a variety of valid elastic behaviors that our model allows. In this section, we list some properties that can be used to better characterize these behaviors. All the properties described in this section focus on resource management, that is, how resources are allocated and deallocated by the system.

Precision identifies how good is the elastic system in allocating and deallocating the right number of resources with respect to variation in the load. In other terms, precision constrains the amount of resources that system is allowed to over- or under-provision. We capture precision by means of parameter ϵ and Formula $\mathcal{P}_{div}$, under the "manageable load" assumption:

$$\mathbf{G}(|R - L| < \epsilon) \qquad (\mathcal{P}_{div})$$

The formula[5] states that during the course of system execution the overall difference between the load and the resources allocation cannot differ more than the specified amount ϵ. This parameter should be defined by the designer of the cloud-based application depending on its requirements.

Oscillation. An elastic system that repeatedly allocates and deallocates resources even when the load stays stable is said to *oscillate*. Oscillations may appear as a consequence of the discrete nature of resources allocation in combination with poorly-designed conditions that trigger adaptation. For example, an elastic controller might try to allocate an average capacity of 1.5 virtual machines by switching between the allocation of one virtual machine and two virtual machines. Despite oscillations being a valid elastic behavior, they might impact on the running costs of the system. We characterize *non-oscillating* behaviors with the following formulae:

$$\mathbf{G}(\mathsf{X}(R) > R \rightarrow \mathbf{P}_{(0,T_e]}(\mathsf{X}(L) > L)) \qquad (\mathcal{P}_{po1})$$

$$\mathbf{G}(\mathsf{X}(R) < R \rightarrow \mathbf{P}_{(0,T_e]}(\mathsf{X}(L) < L)) \qquad (\mathcal{P}_{po2})$$

The formulae constrain the increase (decrease) of the number of resources only in correspondence with an increase, respectively an decrease, of the load. The formulae use the eagerness parameter (T_e) to limit the observation of load variations in the past (expressed with $\mathbf{P}_{(0,T_e]}$). If the controller performs an adaptation not "justified" by a change in the load, it will violate the property captured by the formulae above.

Resource Thrashing. Elastic systems may present opposite adaptations in a very short time; for example, a system may scale up, and then, right after finishing the adap-

[5]For clarity, in the formula we use the metric $|\cdot|$, which does not belong to the CLTL$^t(\mathcal{D})$ syntax. A CLTL$^t(\mathcal{D})$ compliant formulation can be obtained by applying the following rule: $|a| \sim b \equiv (a \sim b \wedge a \geq 0) \vee (-a \sim b \wedge a < 0)$, where $\sim$ is a relational operator.

tation, it can start to scale down. This situation is commonly known as *resource thrashing*. In other words, resource thrashing is a temporary, yet very quick, oscillation in the allocation of resources. In the case of a resource thrashing situation, the resources that are impacted by the adaptation generally do not perform any useful work, yet they contribute to an increment of the running costs. Resource thrashing is parametrized by a minimum time T_{rtx} allowed between an increase and a decrease in the number resources. This time is usually defined by the designer of the cloud-based application, after taking into account the actuation delay of the controller. For a system *not* manifesting a resource trashing condition, the following formulae should hold:

$$\mathbf{G}(R < \mathsf{X}(R) \rightarrow \neg\mathbf{F}_{(0,T_{rtx}]}(R > \mathsf{X}(R))) \qquad (\mathcal{P}_{rtx1})$$

$$\mathbf{G}(R > \mathsf{X}(R) \rightarrow \neg\mathbf{F}_{(0,T_{rtx}]}(R < \mathsf{X}(R))) \qquad (\mathcal{P}_{rtx2})$$

The formulae constrain the occurrence of opposite adaptations to happen after a minimum amount of time T_{rtx}.

Cool-down period is a strategy adopted by designers to achieve a bounded number adaptations over a period of time. It is used to prevent the controller from adapting faster than the time needed for the actual actuation on the cloud-based system. The controller is required to *freeze* for a given amount of time and let the system stabilize after an adaptation. We consider a system unstable if it is in the process of adaptation; this is represented by proposition A. In the following formulae

$$\mathbf{G}\left(\forall id : \begin{pmatrix} \neg M_{ready}(id)\ \mathbf{S}\ M_{start}(id) \\ \vee \\ \neg M_{end}(id)\ \mathbf{S}\ M_{stop}(id) \end{pmatrix} \rightarrow A\right) \qquad (\mathcal{M}_{ai})$$

$$\mathbf{G}\left(\exists id : A \rightarrow \begin{pmatrix} \neg M_{ready}(id)\ \mathbf{S}\ M_{start}(id) \\ \vee \\ \neg M_{end}(id)\ \mathbf{S}\ M_{stop}(id) \end{pmatrix}\right) \qquad (\mathcal{M}_{adp})$$

we yield the proposition A true whenever an adaptation is currently in progress: either event $M_{start}(id)$ or $M_{stop}(id)$ were issued, but no M_{ready} (respectively M_{end}) event is observed. This notions are expressed using the *"Since"* (**S**) modality. We can then use proposition A to express the fact that the controller needs to wait for all recently allocated resources to be ready before performing a new adaptation. This can be represented as a constraint on R to not change when A holds:

$$\mathbf{G}(A \rightarrow \mathsf{Y}(R) = R) \qquad (\mathcal{P}_{cdp})$$

Bounded concurrent adaptations. Sometimes forcing the controller not to react during adaption can be considered a very rigid policy. We can relax this requirement by allowing the controller a fixed number of actions during the adaptation. This property can be viewed as a generalization of the previous one, where the fixed number of actions during adaption was one. To formalize this property we rely on formulae $\mathcal{M}_{ai}$ and $\mathcal{M}_{adp}$ to distinguish the time positions during which an adaptation of the system occurs. The constant M_a represents the maximum number of allowed actions for the controller (either allocations or deallocations) while the system is in the unstable state. In the formalization, we also introduce an additional variable c_a that counts how many overlapping adaptations occur.

$$c_a = 0 \qquad (\mathcal{P}_{bca1})$$

$$\mathbf{G}((\mathsf{Y}(R) \neq R) \rightarrow \mathsf{X}(c_a) = c_a + 1) \qquad (\mathcal{P}_{bca2})$$

$$\mathbf{G}((A \wedge \mathsf{Y}(R) = R) \rightarrow \mathsf{X}(c_a) = c_a) \qquad (\mathcal{P}_{bca3})$$

$$\mathbf{G}(\neg A \rightarrow \mathsf{X}(c_a) = 0) \qquad (\mathcal{P}_{cdp3})$$

$$\mathbf{G}(c_a < M_a) \qquad (\mathcal{P}_{bca4})$$

Formula ($\mathcal{P}_{bca1}$) initializes the variable c_a at position 0. Formulae ($\mathcal{P}_{bca2}$) and ($\mathcal{P}_{bca3}$) update c_a. Formula ($\mathcal{P}_{bca2}$) increases c_a when there is a change in the number of resources ($\mathsf{Y}(R) \neq R$), while formula ($\mathcal{P}_{bca3}$) propagates the current value of c_a during adaptation (hence, A is required to hold in its antecedent). When the system is not adapting (denoted by $\neg A$) we reset the value of c_a to zero, expressed in ($\mathcal{P}_{cdp3}$). Finally, we constrain the value of c_a to be less then M_a over the whole execution trace.

Bounded resource usage. The running costs of elastic systems can be constrained by specifying properties that apply on the whole set of resources in use. For example, we can specify a constraint on the absolute amount of resources in use by the system, as done in Section 4 with Formula $\mathcal{M}_{bound}$. We can also specify time-dependent constraints that temporarily bound the maximum number of resources to certain predefined levels. Time dependent constraints are useful if the budget allocated to the elastic system is very limited, and one must guarantee that the system will run for a given period of time. If the budget is exhausted while the system is still running, the infrastructure abruptly stops and deallocates all the virtual machines. To avoid this situation, an elastic system might need to limit the use of resources beyond a certain threshold, for a specified time interval. We call this requirement *bounded resource usage* and define two parameters to characterize it: a stricter resource bound R_{tmax} with $R_{tmax} < R_{max}$, that represents the temporary new threshold for allocating resources, and a time bound T_{bru}, within which resources above the threshold R_{tmax} should be released. This requirement is expressed as follows:

$$\mathbf{G}(R > R_{tmax} \rightarrow \mathbf{F}_{(0,T_{bru}]}(R \leq R_{tmax})) \qquad (\mathcal{P}_{bru})$$

The formula states that whenever the controller allocates more resources than allowed by the temporary threshold, it needs to release them within T_{bru} time.

5.3 Quality of Service

As any other computer system, elastic systems must provide some predefined level of QoS; however, cloud-based elastic systems introduce a new way to enforce them with respect to non-elastic systems. In the rest of this section, we describe some properties that determine the QoS perceived for a cloud-based system.

Bounded QoS degradation. Implementing a system adaptation, such as scaling up or down resources, may incur in non-trivial operations inside the system. Component synchronization, registration, data replication and data migration are just the most widely known examples. During systems adaptation it may happen that the system shows a degraded QoS. Elastic systems may be required to limit this amount of QoS degradation.

Assuming that the level of quality of service is measurable with a single value, we model the normally-required

QoS limit with the parameter c. In addition, we define the parameter d to model the reduced (degraded) bound on the value of QoS ($c \geq d$). We formalize the requirements on *bounded degradation during adaptation* as follows:

$$\mathbf{G}(A \rightarrow Q > d) \land \mathbf{G}(\neg A \rightarrow Q > c) \qquad (\mathcal{P}_{bqos})$$

The formula above says that the threshold on the normally-required QoS level c should be satisfied only when the system is not performing any adaptation. Instead, during adaptation, the relaxed value d for the QoS is enforced.

Bounded actuation delay. The performance of an elastic system is greatly impacted by the reaction time of its controller, and a controller with a slow reaction time may determine non-effective elastic systems, because adaptations are triggered too late. However, even though the controller triggers an adaptation in time, the system could still be non-effective if the resources take too long to be ready. For this reason, we constrain the actuation delay of the controller with the following formulae:

$$\forall id : \mathbf{G}(M_{boot}(id) \rightarrow \mathbf{F}_{(0,T_{ad})}(M_{ready}(id))) \qquad (\mathcal{P}_{bad})$$

$$\forall id : \mathbf{G}(M_{stop}(id) \rightarrow \mathbf{F}_{(0,T_{ad})}(M_{end}(id))) \qquad (\mathcal{P}_{bad'})$$

We introduce a temporal bound, denoted by parameter T_{ad}, in Formula ($\mathcal{P}_{bad}$) (respectively, ($\mathcal{P}_{bad'}$)) between occurrences of the $M_{boot}(id)$ and $M_{ready}(id)$ events (respectively, $M_{stop}(id)$ and $M_{end}(id)$). Intuitively, the time needed for the application to be ready to serve requests must be less than T_{ad}.

6. PRELIMINARY EVALUATION

We evaluated our formalization of the relevant properties of cloud-based systems by determining whether it could be effectively applied to check this class of properties over execution traces of realistic applications. In particular, we performed *off-line trace checking*[6] of some of the properties described in Section 5 over system execution traces. Our goal was to evaluate the resource usage (execution time and memory) of the trace checking procedure for the different types of properties.

6.1 Methodology

Traces were obtained from the execution of an instance of the "Elastic Doodle" service (see below), deployed over a private OpenStack cloud infrastructure. To trigger elastic behaviors in the application, we created several input workloads, fluctuating according to sine waves, squared waves, and sawtooth patterns. We configured the monitoring tools of the application to create a set of execution traces, each of them containing the timestamped events corresponding to the allocation and deallocation of virtual machines. We leveraged the AUToCLES tool [6] to automate the execution of multiple runs of the system with different input workloads, and to perform data collection.

The following properties were selected for verification over the generated traces: (RT) Resource thrashing; (PL) Plasticity; (CDP) Cool down period.

We used the ZOT verification toolset[7] to perform the actual trace checking procedure. The ZOT toolset supports satisfiability checking [1] [2] of CLTL$^t(\mathcal{D})$ formulae by means of SMT solvers. We translated the traces collected from the execution of the "Elastic Doodle" service into a CLTL$^t(\mathcal{D})$ formula, where each occurrence of a virtual machine allocation or deallocation event is mapped onto an atomic proposition holding at the corresponding timestamp. The logical conjunction of this formula, of the formulae presented in Section 4, and of the CLTL$^t(\mathcal{D})$ version of each of the properties to verify was then provided as input to ZOT. For each verification run, we recorded the memory usage and the SMT verification time.

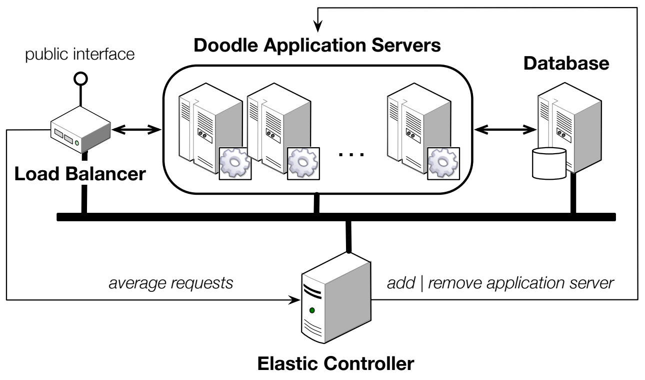

Figure 3: High-level architecture of the "Elastic Doodle" service.

The Elastic Doodle

The "Elastic Doodle" service is an open-source clone of the popular Internet calendar tool of the same name. We extended the original code base to support elastic behaviors, by including both the adaptation logic and the elastic controller. We also added advanced monitor capabilities.

Figure 3 shows the high-level architecture of this elastic service. The system is organized as a n-tier system with a *load-balancer* that exposes the service endpoint to end-users on one side, and forwards client requests to a lineup of *application servers* on the other side. The application servers interact with a shared *database* that acts as the storage/persistency tier of the system. The middle tier has an elastic behavior: instances of the applications server composing this tier can be dynamically added and removed. This elastic behavior is determined by the *controller*, which periodically (e.g., every ten seconds) reads the monitored data and decides on the next resource allocation strategy using a rule-based approach. The following two rules are in place:

scale-up: if the average number of requests per running application server in the last minute is over a certain maximum threshold, a new instance of application server is allocated; the controller stops its execution for one minute;

scale-down: if the average number of requests per running application server in the last minute is below a certain minimum threshold, a running instance of application server is deallocated; the controller stops its execution for two minutes.

[6]Also called *history checking* or *post-mortem analysis*.
[7]`http://code.google.com/p/zot/`.

Table 1: Evaluation data of the Elastic Doodle service

Trace				RT	PL	CDP
ID	Events	Time span (s)	Max resources	Time (s)/Memory(MB)	Time (s)/Memory(MB)	Time (s)/Memory(MB)
T1	15	1102	2	1.44/120.1	1.20/117.7	2.29/126.2
T2	43	635	4	2.83/135.3	1.47/121.8	1.42/121.5
T3	29	641	3	1.77/131.5	1.21/117.7	1.62/126.2
T4	17	499	3	1.20/116.7	1.27/116.0	1.38/115.9
T5	44	644	3	2.94/135.4	1.45/122.1	1.45/121.7

6.2 Results and Discussion

Our trace checking procedure processes the logs generated by the Elastic Doodle service and filters only significant events like M_{start}, M_{boot}, M_{ready}, M_{stop}, M_{end} and any change in the value of the load assigned to L. These events were conjuncted into a formula representing the trace. Let us define a formula $(\mathcal{M}_R)$ as a conjunction of all formulae that define the behavior of the variable R: $(\mathcal{M}_{init}) \wedge (\mathcal{M}_{load}) \wedge (\mathcal{M}_i) \wedge (\mathcal{M}_d) \wedge (\mathcal{M}_{ix}) \wedge (\mathcal{M}_{dx}) \wedge (\mathcal{M}_{eq})$. Similarly, we define $(\mathcal{M}_A)$ as the conjunction of formulae that define the behavior of proposition A: $(\mathcal{M}_{ai}) \wedge (\mathcal{M}_{adp})$.

For the property (RT) we perform bounded satisfiability checking of the formula $\mathcal{F}_{RT}= (\mathcal{M}_R) \wedge \neg\, ((\mathcal{P}_{rtx1}) \wedge (\mathcal{P}_{rtx2}))$ over the traces. We conjunct the term $(\mathcal{M}_R)$ in $\mathcal{F}_{RT}$ because resource thrashing formula relies on variable R. We choose T_{rtx} to be 50 seconds. For the property (PL) we check formula $\mathcal{F}_{PL}=(\mathcal{M}_R) \wedge \neg\, (\mathcal{P}_{pl})$. Plasticity formula also uses variable R, hence the conjunct $(\mathcal{M}_R)$. We choose T_{p_1} to be 3 minutes and T_{p_2} 30 seconds. Finally, for the property (CDP) we check $\mathcal{F}_{CDP}=(\mathcal{M}_R) \wedge (\mathcal{M}_A) \wedge \neg\, (\mathcal{P}_{cdp})$. Since cool down period formula relies both on variable R and proposition A we conjunct their definitions with $\mathcal{F}_{CDP}$.

Notice that in $\mathcal{F}_{RT}$, $\mathcal{F}_{PL}$ and $\mathcal{F}_{CDP}$ we are negating the property formulae. This is done because we perform satisfiability checking and the models of the formulae in fact represent counterexamples of the properties.

We chose five traces with different values for the number of significant events, the time span, and the maximum number of resources being allocated during execution. We report time and memory results for the verification of each property RT, PL and CDP in Table 1. Besides the size of the formula, the parameters that affect the time and memory of the verification procedure are the number of significant events in the traces and the number of resources allocated during the execution.

The trace checking procedure confirmed that traces T1, T3 and T5 satisfy all three properties. Trace T2 violates property RT due to deallocation of a resource 30 seconds after its allocation; it also violates property PL since not all resources are deallocated in the last 3 minutes. Trace T4 violates both properties RT and CDP because of a single (wrong) decision made by the elastic controller: it deallocated a resource while another resource was still initializing, within 50 seconds.

The results of the evaluation suggest that checking the proposed properties formalized in CLTL$^t(\mathcal{D})$ over realistic execution traces is feasible, since the time needed for executing the checking is small (1.66 seconds on average) and the amount of memory required is reasonable (123MB on average).

7. RELATED WORK

There have been few proposals for modeling and formalizing elastic properties in the literature so far. Herbst et al. [7] highlight the need for a precise definition of elasticity in the context of cloud computing. Similarly to the modeling approach followed in this paper, the authors characterize the degree of elasticity in terms of speed of adaptation and precision of adaptation. Starting from basic concepts such as adaptation, demands and capacity the authors define a set of properties to describe the elastic behavior of cloud-based systems. However, these descriptions are informal; the paper only described a set of metrics for measuring system elasticity.

Islam et al. [8] provide a quantitative definition of elasticity using financial terms, taking the point of view of a customer of an elastic system who wants to measure the elasticity provided by the system. The authors measure the financial penalty for systems under-provisioning (due to SLA violations) and over-provisioning (unnecessarily costs) using a reference benchmark suite to characterize the system elasticity. Several critical situations identified in [8] have been reported/discussed in this paper. However, Islam and coauthors provide only an informal description for the properties.

A formal definition and modeling of system plasticity is provided in [5]. The authors model elastic systems by means of state transition systems where transitions are associated with probabilities of switching between states, i.e., different resources allocations, as they are observed in the system run. Plasticity is identified when the model has transitions corresponding to scaling up but lacks (some) transitions corresponding to scaling down. The authors use the proposed model to define an automated procedure for the generation of test cases that expose plastic behaviors of cloud-based elastic systems.

8. CONCLUSION AND FUTURE WORK

In this paper we have shown a possible formalization of cloud-based elastic systems, based on the CLTL$^t(\mathcal{D})$ temporal logic. The concepts and properties related to the behavior of cloud-based elastic systems have been derived from our experience in research projects related to cloud computing and are commonly encountered when designing a cloud-based elastic system. By formalizing the properties using a temporal logic, we are able to leverage automated verification tools for checking whether the proposed properties hold or not during the execution of cloud-based elastic systems. Indeed, we have also reported on our preliminary evaluation showing that it is feasible to check these properties over execution traces of a realistic system, using an SMT-based verification toolkit.

This paper marks only the initial step of our work on the formalization, specification, and verification of properties of cloud-based elastic systems. We are currently working on the refinement of the proposed modeling and formalization to better represent the load of the system and the impact of the the number of resources on the load at a certain time. We are also working on the extension of the proposed modeling to account for other elasticity mechanisms different from horizontal scalability (i.e., virtual machine replication). More specifically, we plan to consider vertical scaling (to dynamically change the amount of resources allocated to single instances of a virtual machine), migration of virtual machines (to enable re-balancing of the load in the cloud), and software and (virtual) hardware reconfigurations. We plan to conduct a thorough evaluation of the performance of checking the proposed properties using execution traces derived from industrial-strength case studies. Furthermore, we plan to integrate the underlying trace checking approach into a run-time monitoring (and verification) platform for cloud-based elastic systems.

9. ACKNOWLEDGMENTS

This work has been supported by the National Research Fund, Luxembourg (FNR/P10/03); by the Swiss National Science Foundation under the "Fellowship for Prospective Researches" contract PBTIP2-142337; by the Austrian Science Fund (FWF) under project reference P23313-N23 (Audit4SOAs).

10. REFERENCES

[1] M. M. Bersani, D. Bianculli, C. Ghezzi, S. Krstić, and P. San Pietro. SMT-based checking of SOLOIST over sparse traces. In *Proc. of FASE 2014*, volume 8411 of *LNCS*. Springer, April 2014.

[2] M. M. Bersani, A. Frigeri, A. Morzenti, M. Pradella, M. Rossi, and P. San Pietro. Constraint LTL Satisfiability Checking without Automata. *CoRR*, abs/1205.0946v2, 2013.

[3] CELAR Project. Description of Work. `http://www.celarcloud.eu`, 2012.

[4] S. Demri and D. D'Souza. An automata-theoretic approach to constraint LTL. *Inf. Comput.*, 205(3):380–415, 2007.

[5] A. Gambi, A. Filieri, and S. Dustdar. Iterative test suites refinement for elastic computing systems. In *Proc. of ESEC/SIGSOFT FSE 2013*, pages 635–638. ACM, 2013.

[6] A. Gambi, W. Hummer, and S. Dustdar. Automated testing of cloud-based elastic systems with AUToCLES. In *Proc. of ASE 2013*, pages 714–717. IEEE, 2013.

[7] N. Herbst, S. Kounev, and R. Reussner. Elasticity in cloud computing: What it is, and what it is not. In *Proc. of ICAC 2013*, pages 23–27. USENIX, 2013.

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

[9] S. K. Lahiri and M. Musuvathi. An efficient decision procedure for UTVPI constraints. In *Proc. of FroCoS 2005*, volume 3717 of *LNAI*, pages 168–183. Springer, 2005.

[10] O. Lichtenstein, A. Pnueli, and L. Zuck. The glory of the past. In *Proc. of Logics of Programs*, volume 193 of *LNCS*, pages 196–218. Springer, 1985.

[11] P. Mell and T. Grace. The NIST Definition of Cloud Computing. `http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf`, September 2011. NIST Special Publication 800-145.

[12] S. Naqvi and P. Massonet. RESERVOIR - a european cloud computing project. *ERCIM News*, 2010(83):35, 2010.

[13] M. Pradella, A. Morzenti, and P. San Pietro. Bounded satisfiability checking of metric temporal logic specifications. *ACM Trans. Softw. Eng. Methodol.*, 22(3):20:1–20:54, July 2013.