2015 IEEE/ACM 7th International Workshop on Principles of Engineering Service-Oriented and Cloud Systems

A Pattern-Based Formalization of Cloud-Based Elastic Systems

Schahram Dustdar*, Alessio Gambi*, Willibald Krenn† and Dejan Nickovic†
*Vienna University of Technology, †Austrian Institute of Technology
Vienna, Austria
{name.surname}@tuwien.ac.at, {name.surname}@ait.ac.at

***Abstract*—Cloud-based elastic systems leverage cloud infrastructures to implement elasticity, the ability of computing systems to dynamically adjust their capacity by changing the allocation of resources in response to fluctuating workloads. The runtime behavior of elastic systems is the result of an intricate interplay of many factors that include the input workload, the elasticity logic determining the resources allocation, and the technology of the underlying cloud. This makes elastic systems difficult to design and hard to specify.**

In this paper we propose a novel formalization of elasticity and related concepts that is based on timed patterns written using timed regular expressions. Timed regular expressions naturally deal with dense-time signals, and timed patterns allow us to intuitively describe relevant changes in those signals. This, in turn, enables us to directly characterize elasticity as relation between relevant changes in the input workload and in the resources allocation signals. We firstly characterize the relevant changes by means of timed patterns, and then we define desired and undesired behaviors of cloud-based elastic systems in terms of the occurrence of such patterns over an observation period.

I. Introduction

The advent of cloud computing gave raise to a new breed of self-adaptive systems called cloud-based elastic systems. Cloud-based elastic systems leverage the ability of cloud platforms to deliver computing resources on-demand as remote services for implementing elasticity, the ability to dynamically scale by acquiring and releasing computing resources.

By dynamically acting upon resources allocation elasticity can control the available system capacity at run time. Application providers exploit this fact to avoid under- and over-provisioning of their cloud-based systems when the workloads entering their systems fluctuate. Indeed application providers implement elastic systems with the aim of maintaining consistent levels of service while minimizing the running costs by adjusting the provided capacity to match the capacity required by the fluctuating workloads.

The runtime behavior of cloud-based elastic systems results from an intricate interplay of several factors that makes the design of elastic systems very challenging: the input workload that fluctuates; the cloud platform that delivers the computing resources; the way the applications deployed on the cloud adapt; and, the logic that decides on the allocation of resources.

If not properly designed elasticity might turn into a double-edged sword and possibly cause undesired emergent behaviors, such as instability and resource thrashing, that might jeopardize the system dependability and the business of application owners [1]. This calls for systematic methodologies to develop elastic systems as well as approaches that allow application providers to precisely characterize the desired behaviors of their elastic systems.

In this paper, we address the latter point and propose a novel formalization of the behavior of cloud-based elastic systems based on timed regular expressions. Timed regular expressions naturally deal with both continuos and integer dense-time signals and allow application providers to directly express, as timed patterns, the relevant aspects of the evolution of systems variables, such as input workload, system performance and resources allocation. This way, application providers can formalize elasticity more intuitively by describing *how* the various signals, i.e., the system variables, should evolve and co-evolve. For example, using timed regular expressions it comes natural to describe what constitute a relevant increase of the workload, e.g, in terms of its first derivative, and correlate such an event to an expected increase of the resources allocation.

Our work advances the current state of research by re-formulating common elasticity related properties using timed regular expression, and by introducing additional properties to describe the expected functioning of the inner components that comprise cloud-based elastic systems. In other words, instead of limiting the scope of our formalization at the outer *system* level, we whiten the model of cloud-based elastic systems and formalize their behavior at *component integration* level.

The rest of the paper is organized as follows. Section II introduces the reference architecture for cloud-based elastic systems. Section III summarizes the main elements of timed regular expressions. Section IV presents our approach to formalize elasticity using patterns written as timed regular expressions. Section V extends the proposed formalization to include threshold-based elastic controllers. Section VI discusses the related work, and Section VII concludes the paper.

II. Cloud-based Elastic Systems

The key aspect of elastic systems is their ability to self-adapt at run time in response to changes in the operating conditions, such as seasonal fluctuations of the number of input requests, by suitably *stretching* and *shrinking*.

Cloud-based elastic systems are the most common implementation of elastic systems. They leverage the ability of the cloud infrastructures to dynamically allocate and deallocate

2156-793X/15 $31.00 © 2015 IEEE
DOI 10.1109/PESOS.2015.13

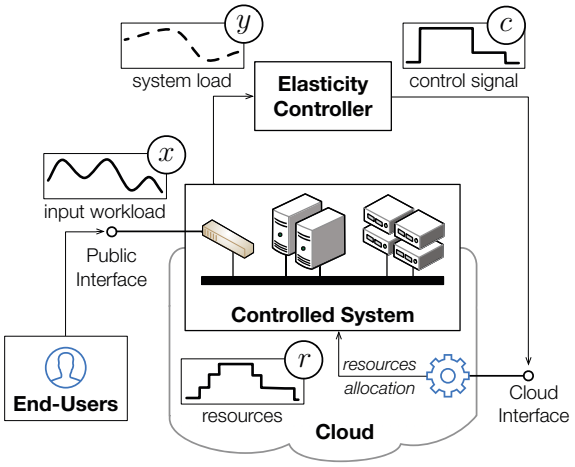

Fig. 1. Reference architecture of cloud-based elastic systems

computing resources to stretch and shrink. For example, when the workload increases the allocated computing resources might saturate and in order to avoid degradation of the Quality of Service (QoS) elastic systems acquire additional resources over which distribute the load. Hence, the elastic systems *scale up*. When the incoming workload decreases and the allocated computing resources become under-utilized, the elastic system will reduce costs by consolidating the system load on a portion of the allocated resources and release the unused ones. The elastic systems *scale down*. This way, elasticity is a means to avoid under- and over-provisioning, and allows elastic systems to maintain a suitable QoS while minimizing the running costs. This, in turn, contributes to the satisfaction of both end-users and application providers.

As many other self-adaptive systems, cloud-based elastic systems implement a closed-loop architecture [2]. Figure 1 depicts the reference architecture for cloud-based elastic systems, where an *elasticity controller* supervises a *controlled system*, i.e., the actual system that implements the business logic of the application. *End-users* access the controlled system through its public interface and generate the *input workload* – x. The *controlled system* consists of a set of virtual machines that are deployed onto a cloud and cooperate to respond to end-user requests. The controlled system also implements the logic that adjusts the capacity of the system when the allocation of resources changes, which is required to enable elasticity. The elasticity controller monitors the operating parameters of the controlled system (e.g., the *system load* – y) and periodically determines the control actions to be executed to perform adaptation (cf. the *control signal* – c). Notice that signal y is multi-dimensional and might include x.

Based on the control signal the cloud dynamically instantiate and terminate virtual machines. Notice that the process of instantiating/terminating virtual machines might take a non-negligible amount of time. Finally, the cloud tracks the total resource usage of the controlled system over time (cf. *resources* – r) and bills the application provider for the cost of running the elastic system.

III. Basics of Timed Regular Expressions

Timed regular expressions are an extension of traditional regular expressions that enables specifying real-time patterns over dense-time signals [3]. By means of timed regular expressions, we can provide formal specifications that concisely characterize the behavior of systems in terms of co-evolving mixed, integer and continuous valued signals and instantaneous events.

Each timed regular expression captures a well defined and localized evolution of a set of signals into a pattern. Basic patterns can be flexibly composed to obtain bigger patterns that describe complex situations by means of traditional operators like concatenation ($\cdot$), repetition (*), union ($\cup$) and intersection ($\cap$).

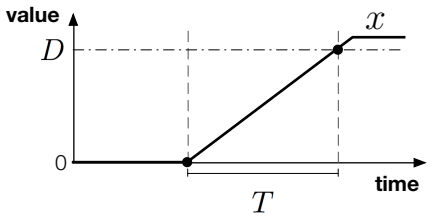

Fig. 2. The ramp up pattern

We illustrate the specification of patterns over dense-time signals with the following pedagogical example. Consider a real-valued signal x that at some point in time *ramps up* as shown in Figure 2. Intuitively, this behavior can be captured by a composed pattern that identifies where x changes, describes each of the segments comprising x, and then concatenates the segments. In particular, before the ramp begins the value of x must be stable and equal to 0, after the ramp the value of x must be at least equals to a target (positive) value D, and during the ramp the value of x must be always between 0 and D while non-monotonically increasing. Furthermore, the ramp must have a bounded duration T.

We match timed regular expressions on *finite signals* of duration d. We consider Boolean, integer and real valued signals. The basic block in a timed regular expression is a *propositional term* p, that is either a Boolean variable b, a predicate $\theta(x)$ over an integer or a real variable x, or their negation $\neg b$ and $\neg\theta(x)$ respectively.[1] The syntax of timed regular expressions is defined according to the following grammar:

$$\varphi := \epsilon \mid p \mid \uparrow p \mid \varphi_1 \cdot \varphi_2 \mid \varphi_1 \cup \varphi_2 \mid \varphi_1 \cap \varphi_2 \mid \varphi^* \mid \langle\varphi\rangle_J$$

where p is a propositional term and J is a rational non-empty interval. Timed regular expressions contain all the classical elements of regular expressions, such as the empty word ϵ, the

[1]We will abuse notation and also directly use $\neg p$ to refer to a propositional term that is the negation of a Boolean variable or a predicate.

propositional term p, the traditional set-theoretic operators, and specific operators, like the timed restriction operation $\langle\varphi\rangle_J$, which allows us to reason about patterns of length determined by the interval J. In addition, we allow rising events $\uparrow$ of propositional terms ($\uparrow p$).

Intuitively, the semantics of a timed regular expression relate an expression φ to the signal segments that match it. For instance, only empty signal segments match ϵ, while a signal segment matches p ($\neg p$) if p is true (false) throughout the segment. The rising event $\uparrow p$ is matched only by singular points in the signal at which p goes from false to true value. A signal segment matches $\langle\varphi\rangle_J$ if it matches φ and the duration of the segment is contained in the interval J. All the other operators behave as in the classical regular expressions.

In the following we use the notation $\circ_{i=1}^{n}\varphi_i$ as syntactic sugar for $\varphi_1 \cdot \varphi_2 \cdots \varphi_n$. The fall event $\downarrow p$ can be obtained as syntactic sugar $\uparrow \neg p$, the propositional constant true $\top$ as $p \cup \neg p$ and $\bot$ as $p \cap \neg p$. Positive repetition is written φ^+ and is equivalent to $\varphi \cdot \varphi^*$.

Given the syntax of timed regular expressions we can formalize the ramp up pattern:

$$(x = 0) \cdot \langle x \in (0, D) \cap \dot{x} \geq 0\rangle_{\leq T} \cdot (x \geq D)$$

where $\dot{x}$ denotes the first derivative of x.[2]

We derive the specifications of systems under development by refining the timed patterns, providing values to all the parameters (i.e., D and T in the example), and introducing additional constraints about the occurrence (or absence) of the timed patterns. Timed pattern matching [4] is the problem of effectively computing all the segments in a dense-time signal that match a timed regular expression. Computing all such segments enables versatile analysis of real-time behaviors. For instance, one can detect and visualize the presence or the absence of desired or undesired behaviors, count the number of times timed patterns occur in the signal, derive the durations of the matched segments, etc. This approach provides richer analysis of real-time behaviors than the classical monitoring based on temporal logic [5], [6].

Dense-time signals, as well as instantaneous events, naturally arise in cloud-based elastic systems where monitored variables such as input workload, performance, reliability metrics and resources allocation can be described as signals, while the life-cycle of virtual machines can modeled in terms of instantaneous events [6]. Therefore, we argue that timed regular expressions are a suitable formalism for describing the behavior of such systems and characterizing their elasticity. In fact, timed regular expressions naturally deal with signals and events, and, as we illustrate in the next sections, enable the designers to decompose complex elasticity properties into a set of smaller, more intuitive, timed patterns that when combined can describe the intended adaptations of the elastic systems in the face of the ever-changing operating conditions.

[2]We assume that the derivative $\dot{x}$ can be computed or approximated from x. Notice that this formula is rather abstract and allows for many different "shapes" of the ramp up, which are not limited to the linear case depicted in Figure 2.

IV. A Pattern-based Specification of Elasticity

Elastic systems dynamically allocate resources in response to the changes in the input workload. In the case of cloud-based elastic systems, this adaptation results either in an increase or a decrease of the computing capacity, by allocation or release of the resources. While the decisions about allocation and de-allocation of resources come from the elastic controller, the elasticity properties of the system can be observed at the system interface, as relations between the input workload x and the amount of resources allocated r. Reasoning about cloud-based elastic systems at this level of abstraction allows us to specify and observe both desired and undesired elasticity related properties at the system level, without the need to refer to the actual implementation of its components.

In this paper, we propose to capture relevant variations of x and r by means of timed patterns, compose the patterns to describe how r should change given that x is evolving, and eventually derive a set of formal specifications that capture the expected elasticity related properties in terms of the occurrence of the corresponding patterns in the observation period. Intuitively, universal properties must match across the whole observation period, and expected properties must match the intended amount of times and possibly for the indented duration. Consequently, undesired properties, which are a type of expected properties, must match no times over the observation period, meaning that their occurrence is never observed. For example, elastic systems should always adapt as within a given time after the input workload changes enough; conversely, the systems should never adapt if the input workload does not change.

We decide to use a compositional approach based on timed patterns over dense-time signals for several reasons. Timed patterns have localized nature that makes them intuitive. In fact, the patterns focus on describing only specific segments of signals, for example where a signal increases over a threshold; nevertheless, they can span across wide intervals, theoretically unbounded. Timed patterns are composable, concise and declarative. Complex patterns can be formed by combining simpler patterns using for example sequential operations and repetitions. This enables us to describe complex situations that possibly involve multiple signals in high-level terms by abstracting several low level details. Furthermore, patterns promote reuse. For example, with timed patterns is easy to write a specification that prohibits an elastic system to scale up more than two times in each minute. We can achieve this by firstly describing a generic scale up in terms of increase of the resources allocation; concatenating two consecutive scale up patterns and restricting them to appear within a minute; and, finally imposing that such composed patterns must never be matched.

In the remaining of this section, we illustrate our pattern-based formalization of elasticity by presenting the formalization of some of the properties originally introduced by Bersani and coauthors in [6]. We model the input workload x as a continuous-time real-valued signal and the resource

allocation r as a piecewise-constant, non-negative and integer-valued signal. In particular, we require that r is bounded by a minimal and a maximal amount of resources, $r_{min} = \min$ and $r_{max} = \max$, such that $\max \geq \min$. We denote by r_i, where $i \in [\min, \max]$, the amount i of resources. Notice that bounding of r is a common strategy that application providers adopt to guarantee that their systems have sufficient resources to operate and an upper limit on the amount of resources concurrently in use. We capture this **bounded resources** property with the following br pattern expressed on r:

$$\mathsf{br} := \langle r \geq r_{min} \cap r \leq r_{max} \rangle_{>0}$$

To formalize common desirable and undesirable patterns that characterize elasticity in cloud-based systems, we partition the input workload into $r_{max} - r_{min} + 1$ possible value ranges $I_{min}, \ldots, I_{max}$ and associate to each value range I_i the expected amount of allocated resources r_i. Figure 3 illustrates this strategy.

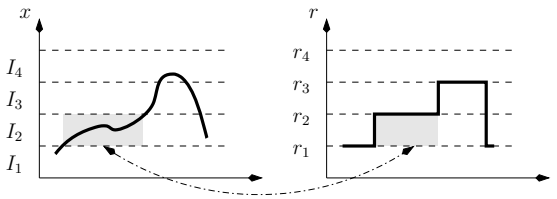

Fig. 3. Partitions of input workload x and their relation to the allocated resources r.

Elastic systems adapt by allocating resources, i.e., they scale up, and deallocating resource, i.e., they scale down; therefore, we characterize the possible adaptations of elastic systems by introducing patterns that describe the increase and, respectively, the decrease of r.

In details, we define a **scale up** between any two amounts of resources, r_i and r_j such that $r_{min} \leq r_i < r_j \leq r_{max}$, as a *monotonically* increase in r. Additionally, we require that scaling up has a maximum duration of t_{su}. We characterize the pattern of scaling up from r_i to r_j, and its arbitrary level scale-up generalization as follows:

$$\begin{array}{lcl} \mathsf{su}(i,j) & := & (r = r_i) \cdot \langle \circ_{k=i+1}^{j-1} (r = r_k)^* \rangle_{\leq t_{su}} \cdot (r = r_j) \\ \mathsf{su} & := & \bigcup_{min \leq i < j \leq max} \mathsf{su}(i,j) \end{array}$$

We omit the formal characterization of scale-down patterns as it is symmetric to the scale up case. We define $\mathsf{su}(i,*)$ and $\mathsf{sd}(i,*)$ as syntactic sugar for $\bigcup_{min \leq i < j \leq max} \mathsf{su}(i,j)$ and $\bigcup_{min \leq k < i \leq max} \mathsf{sd}(i,k)$. Similarly, we define $\mathsf{su}(*,j)$ and $\mathsf{sd}(*,j)$.

The mapping between x and r conveniently captures the requirements about the intended resources allocation with respect to the input workload that the implementation of elastic cloud-based systems must provide. Given this mapping we characterize a system to be elastic if whenever the value of the input workload is continuously within a partition I_i for some time t_w, then the amount of allocated resources becomes equal to r_i within a time t_r. Figure 4 exemplifies this **expected elasticity** concept that we formalize as el pattern as follows:

$$\begin{array}{lcl} \mathsf{el}(i) & := & (x \notin I_i \cap r \neq r_i) \cdot \langle x \in I_i \rangle_{t_w} \cdot \\ & & \langle \mathsf{su}(*,i) \cup \mathsf{sd}(*,i) \rangle_{\leq t_r} \\ \mathsf{el} & := & \bigcup_{min \leq i \leq max} \mathsf{el}(i) \end{array}$$

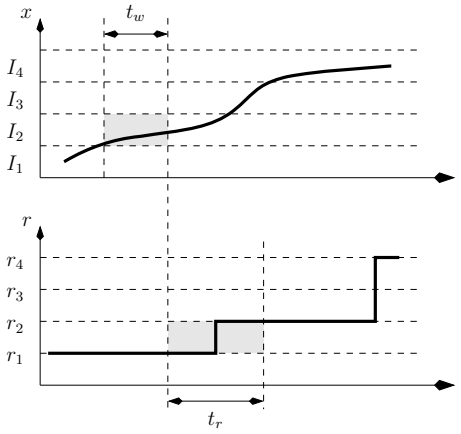

Fig. 4. Example of expected elasticity - illustration of $\mathsf{el}(2)$.

The el pattern allows us to capture all the segments in x and r that satisfy the expected elasticity property. However, it would be also useful to capture those segments where such elasticity property is violated, for example, to identify if critical situations and emergent behaviors arise [1].

In general, the expected elasticity property is violated when x has continuously its value in I_i for t_w time, but the amount of allocated resources fails to reach r_i within t_r time. We formalize this **failed expected elasticity** property as nel pattern as follows:

$$\begin{array}{lcl} \mathsf{nel}(i) & := & (x \notin I_i \cap r \neq r_i) \cdot \langle x \in I_i \rangle_{t_w} \cdot \langle r \neq r_i \rangle_{>t_r} \\ \mathsf{nel} & := & \bigcup_{min \leq i \leq max} \mathsf{nel}(i) \end{array}$$

The nel pattern is rather abstract and can be used to identify all the possible cases where elastic systems do not adapt as expected. Therefore, in order to identify more specific types of violation of the elasticity condition, such as *plasticity* and *inelasticity* [7], we introduce additional patterns that refine nel.

Plasticity is defined as the inability of the system that operates at a given time with r_i resources to adapt back to r_i resources after scaling up or down [8]. We notice that a plastic system can be described as an elasticity system that takes infinite time to complete an adaptation; hence, we formalize this behavior with the following pl patterns:

$$\begin{array}{lcl} \mathsf{pl}_{su}(i) & := & \mathsf{su}(i,*) \cdot \langle x \in I_i \rangle_{t_w} \cdot \langle r > r_i \rangle_{>t_r} \\ \mathsf{pl}_{sd}(i) & := & \mathsf{sd}(i,*) \cdot \langle x \in I_i \rangle_{t_w} \cdot \langle r < r_i \rangle_{>t_r} \\ \mathsf{pl}(i) & := & \mathsf{pl}_{su}(i) \cup \mathsf{pl}_{sd}(i) \\ \mathsf{pl} & := & \bigcup_{min \leq i \leq max} \mathsf{pl}(i) \end{array}$$

Notice that we characterize the behavior of plastic systems by composing patterns that separately describe the plastic case after a scale up (pl_{su}) and the case after a scale down (pl_{sd}).

Inelasticity is a stronger form of violation of the elasticity property and is defined as the total lack of system adaption after variations of the input workload that would require either to scale up or to scale down. We formalize inelasticity with the inel pattern that correlates the input workload and the resource signals as follows:

$$\begin{array}{lcl} \mathsf{inel}(i) & := & (r = r_i \cap x \in I_i) \cdot \langle x \notin I_i \rangle_{t_w} \cdot \langle r = r_i \rangle_{>t_r} \\ \mathsf{inel} & := & \bigcup_{min \leq i \leq max} \mathsf{inel}(i) \end{array}$$

As discussed in [6], elasticity does not prevent cloud-based elastic systems to exhibit behaviors that, despite being formally correct, are undesired from the application provider point of view. For examples behaviors such as *oscillations* and *resource thrashing* might be valid manifestations of elasticity; nevertheless, they should be identified and possibly corrected.

We say that a cloud-based elastic system oscillates when in the face of a stable input workload the system still triggers (possibly multiple) changes in the allocation of resources. For example, oscillations can happen when the value of the input workload signal is near the boundary between two consecutive partitions and although its rate of change is small, it regularly crosses the boundary. Assuming to have access to the signal $\dot{x}$, which can be easily computed by pre-processing x, we formalize **oscillations** with the osc patterns by bounding the value of $\dot{x}$ over the time t_w by a tolerance ε and still observing oscillations in the amount of allocated resources within some time bound t_o:

$$\begin{array}{lcl} \mathsf{osc}(i) & := & (r = r_i) \cdot \langle \dot{x} \in (-\varepsilon, \varepsilon) \rangle_{t_w} \cdot \\ & & \langle (\top \cdot r \neq r_i \cdot r = r_i)^+ \rangle_{\leq t_o} \\ \mathsf{osc} & := & \bigcup_{min \leq i \leq max} \mathsf{osc}(i) \end{array}$$

Resource thrashing can be intuitively described as a special type of oscillation that consists in opposite adaptations occurring in a short period of time that we denote by t_{rt}. We formalize **resource thrashing** with rt patterns by elaborating on the scaling patterns:

$$\begin{array}{lcl} \mathsf{rt}_{su \to sd}(i) & := & \langle \mathsf{su}(*, i) \cdot \mathsf{sd}(i, *) \rangle_{\leq t_{rt}} \\ \mathsf{rt}_{sd \to su}(i) & := & \langle \mathsf{sd}(*, i) \cdot \mathsf{su}(i, *) \rangle_{\leq t_{rt}} \\ \mathsf{rt} & := & \bigcup_{min \leq i \leq max} (\mathsf{rt}_{su \to sd}(i) \cup \mathsf{rt}_{sd \to su}(i)) \end{array}$$

We now illustrate how the pattern-based specification and matching of property work on the example depicted in Figure 5. The figure depicts the case of a fluctuating input workload and the corresponding observed elastic adaptations.

Consider first the case $t_w = 0.5$ and $t_r = 0.25$. In this case, the expected elasticity pattern el is matched twice in the behavior. This is highlighted by the segments s_1 and s_2 in Figure 5. More precisely, the first match corresponds to pattern $\mathsf{el}(2)$, while the second one corresponds to $\mathsf{el}(1)$. Under this settings, there are no matches for patterns $\mathsf{el}(3)$ and $\mathsf{el}(4)$. In fact, the input workload x remains in the region I_3 less than 0.5 time, and hence does not trigger the scaling up of the resource allocation to r_3. If we now change t_w to be equal to 0.25, we obtain different signal segments, namely s_3, s_4 and s_5, that match the failed expected elasticity nel. This is due to the fact that r fails to adapt altogether to the expected allocation of resources (s_3) and fails to adapt in the requested time frame (s_4 and s_5).

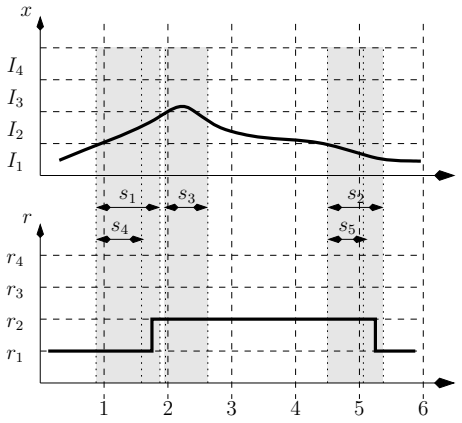

Fig. 5. Example of pattern-based matching of elasticity properties.

The timed patterns presented in this section enable the characterization of elasticity of cloud-based systems in a black-box fashion by elaborating on the input workload, its variations and the resulting system adaptations; however, leveraging these patterns we can formally specify only the outer behavior of cloud-based elastic systems without being able to link the observed behavior to specific components inside them. In the next section we show how we can "open-the-box" and use timed patterns to characterize also the behavior of the components that comprise elastic cloud-based systems. Specifications at the level of component integration allows application providers to link the implementation of systems with their externally observable behavior, therefore, better characterizing the achieved elasticity. We derive such component level specifications by introducing new signals and patterns, and by refining the timed patterns that characterize elasticity.

V. Modeling Threshold-based Elastic Systems

In this section we consider a particular type of elastic systems, namely *threshold-based* elasticity systems [9]. And we provide a formalization of the main properties of their implementation of elastic controllers (see Figure 6). We consider threshold-based elasticity controllers because of their intuitiveness and because they are the "de facto" standard solution for implementing elastic systems in the industry [10].

Threshold-based elasticity controllers monitor a target metric that captures the current load of the system, for example the average CPU usage in the past minute, and produce as output the control signal that drives the resources allocation. To compute the control signal these controllers periodically, with control period P, trigger a simple elasticity logic that demands the acquisition or the release of computing resources when the value of the target metric exceeds respectively the upper or the lower thresholds configured by the application provider.

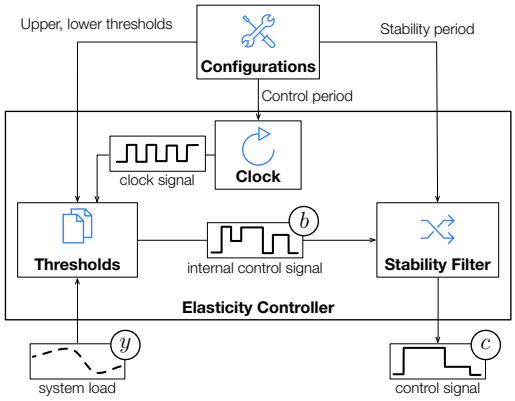

Fig. 6. Details of the threshold-based elasticity controller

Elasticity controllers might decide to temporary disable the triggering of the elasticity logic to avoid to overexcite the controlled systems. This has the effect of filtering away some control actions, thus smoothing the control signal.

We model threshold-based elasticity controllers in terms of three components: the *Clock*, which generates the periodic *clock signal* – *clk* and triggers the elasticity logic; the *Thresholds* component, which samples the *system load* – y at the rising edges of the clock and computes the *internal control signal* – b; and, the *Stability Filter*, which smoothens the internal control signal to produce the actual *control signal* – c. We model the system load as continuous and real-valued signal, and we model the control signals (b and c) as piecewise-constant and integer-valued signals.

We now characterize a number of patterns that are expected to be seen in threshold-based elasticity controllers. We first specify the scaling up pattern in general terms, and then, we provide refinements that account for the possible policies that elasticity controllers can implement for scaling up and down. We denote such scaling policies as SUp and SDp. Note that the scaling up and down patterns are symmetric; therefore we left out from the presentation the scaling down patterns.

Intuitively, a scale up is triggered when the system load crosses the upper bound threshold (denoted as *UB*) and results in a demand for acquiring additional resources. We capture this with the threshold-based scale up pattern (tbSU). tbSU requires that when the value of the signal y at a clock rising edge ($\uparrow clk$) is above the threshold *UB*, then the value of the internal control signal b increases according to the policy SUp:

$$\mathsf{tbSU} \quad := \quad \bigcup_{min \le i \le max}((b = r_i) \cdot (\uparrow clk \cap y > UB \cap \mathsf{SUp}))$$

In its most general form, a valid scale up consists in allocating any amount r_j of resources that is greater than the current amount r_i and that is bounded by r_{max}. We formalize this concept with the SUp_{gen} pattern as follows.

$$\mathsf{SUp}_{gen} \quad := \quad b \geq \min(r_{max}, r_i + 1)$$

In practice however, application providers want to enforce a more precise behavior on their elastic systems; hence, they provide different implementations of the scaling policy [11]. A widely used scaling policy consists in allocating (resp. deallocating) a fixed amount of resources that is always equal to some pre-defined parameter n. [3] We call this policy **linear scaling**. We formalize the functioning of the linear policy with the SUp_{lin} pattern.

$$\mathsf{SUp}_{lin} \quad := \quad b = \min(r_{max}, r_i + n)$$

The linear scaling policy is very intuitive, however, it is inflexible and might limit the effectiveness of using elastic systems. In fact, depending on how the parameter n is chosen, the elastic systems might react very slowly (e.g., small n) or they might over react (e.g., big n). An alternative implementation of scaling policy, which we call **proportional scaling**, computes the next amount of allocated resources as a value that is proportional to the load y modulo the current amount of resource r_i and the thresholds *UB*. We formalize the proportional scaling policy with the SUp_{prop} pattern.

$$\mathsf{SUp}_{prop} \quad := \quad b = \min(r_{max}, \lceil \tfrac{r_i \cdot y}{UB} \rceil)$$

We finally formalize the properties related to the stability filter, which smoothens the internal control signal b into the actual control signal c. The filter can be configured by specifying a stability period S that determines the extent of the smoothing. Without loss of generality, we assume that the stability period is expressed as a multiple m of the control period P, hence $S = mP$.

Given the stability period, the filter computes the value of c periodically at the rising edges of the clock and considers the values of b and c in the previous m control periods as follows: if, after c increased (resp. decreased), b was consistently greater or equal to c (resp. smaller or equal to c) for the whole stability period, then the filter enables the signal c to increase (resp. decrease); otherwise, the filter smoothens the signal and maintains the current value of signal c.

Since the stability period lasts for multiple control periods, elasticity controllers observe the system load y several times before acting. If the load intensity varies, the controllers adopt the conservative strategy to scale the system to cope with the maximal observed load variation. In other words, assume that c was equal to some r_i just before the rising edge of the clock; the value c is the result of a scale up, a scale down or a no-scale operation. At the rising edge of clock c is equal to some r_j such that:

- $r_j < r_i$ if b was always smaller or equal to r_i throughout the last m sampling periods and r_j was the smallest value that b reached during that time;
- $r_j > r_i$ if b was always greater or equal to r_i throughout the last m sampling periods and r_j was the largest value that b reached during that time;

[3] See for example the AutoScaler policy suggested by Amazon: http://docs.aws.amazon.com/AutoScaling/latest/GettingStartedGuide/as-gsg.pdf

- $r_j = r_i$ otherwise.

We formalize these **stability** properties for the scale up operation with the SF_{su} pattern.

$$\begin{array}{lcl} \mathsf{SF}_{su}(i,j) & := & \langle (\top \cdot b = r_j \cdot \top) \cap \\ & & (r_i \leq b \leq r_j) \cap (\top \cdot c = r_i) \rangle_{mP} \cdot \\ & & (\uparrow clk \cap c = r_j) \\ \mathsf{SF}_{su} & := & \bigcup_{min \leq i < j \leq max} \mathsf{SF}_{su}(i,j) \end{array}$$

The stability patterns for the two other cases are similar, therefore we omit them from the exposition.

VI. Related Work

In this paper, we propose a novel formalization of the concept of elasticity in the cloud computing domain. This work complements the other proposals for capturing, measuring, modeling and formalizing elasticity found in literature.

Dustdar et al. [12] introduce the main principles of multi-dimensional elasticity and describe their application in the context of elastic processes. The authors characterize elasticity as multi-dimensional property that describes the various aspects of systems adaptation by correlating different system qualities. In this paper, we adopt a similar philosophy and characterize elasticity as a property that relates input workload, system load, control signal and resource allocation.

In the context of cloud computing, Islam et al. [1] provide a quantitative characterization of elasticity from the point of view of the application providers using the cloud. The authors define elasticity in terms of financial losses caused by under-provisioning and unnecessarily costs due to over-provisioning. Herbst et al. [13] refine the work by Islam and co-authors and characterize elasticity in terms of speed and precision of adaptation, key aspects of dynamic resources allocation. Similarly to these work, we share the main goal of characterize elasticity in quantitative terms; however, differently from them, we aim to provide a precise and formal definition elasticity.

Towards this end, Bersani et al. [6] present a categorization of the most relevant elasticity properties and propose their initial formalization by means of the $\text{CLTL}^t(\mathcal{D})$ temporal logic. In this work, we follow the same categorization, but use a different formalism to model elasticity and related properties. The difference between the satisfaction relation in temporal logics such as $\text{CLTL}^t(\mathcal{D})$ and matching semantics of (timed) regular expressions makes these two classes of formalisms complementary. The satisfaction relation of $\text{CLTL}^t(\mathcal{D})$ is defined relative to single time points, making the temporal logic a good specification language for monitoring applications. On the other hand, a timed regular expression match is defined relative to a pair of points, denoting the start and the end times of successful matches. It follows that timed regular expressions can be naturally used to extract relevant patterns from traces and use them to measure and learn different parameters of the system. Finally, we note that in this paper, we also move a step forward by providing specification of elastic systems at the component integration level.

VII. Conclusions and Future Work

In this paper we have shown a formalization of cloud-based elastic systems based on timed regular expressions and illustrated how timed regular expression can be used to specify the outer behavior of elastic systems, as well as their behavior at the component integration. In particular, we have formalized the behavior of threshold-based elastic controllers, that are the most widely adopted implementation of elastic systems.

Our formalization fosters the use of automated verification tools to monitor the behavior of elastic systems over finite observation periods. This work also enables synthesis of controllers from the specifications and inference of timed patterns from existing controllers. Currently we are working on the first point by extending the implementation from [4] to accommodate the specific needs of automated timed pattern matching for elasticity. For the future, we plan to extend the formalism of timed regular expressions by introducing local variables and measurements. Local variables can capture the state of the system, and their use might simplify the formulation of several elasticity patterns, like su, sd and el. Measurements taken on the matched patterns, instead, will allow us to elegantly introduce quantitative aspects inside the specifications and include constraints about the quality of elastic system adaptation.

References

[1] S. Islam, K. Lee, A. Fekete, and A. Liu, "How a consumer can measure elasticity for cloud platforms," in *Proc. of the Intl. Conf. on Performance Engineering*, ser. ICPE '12, 2012, pp. 85–96.

[2] Y. Brun, G. Di Marzo Serugendo, C. Gacek, H. Giese, H. M. Kienle, M. Litoiu, H. A. Müller, M. Pezzè, and M. Shaw, *Engineering Self-Adaptive Systems through Feedback Loops*. Springer Verlag, 2009.

[3] E. Asaring, P. Caspi, and O. Maler, "Timed regular expressions," *Journal of the ACM*, vol. 49, no. 2, pp. 172–206, Mar. 2002.

[4] D. Ulus, T. Ferrère, E. Asarin, and O. Maler, "Timed pattern matching," in *Formal Modeling and Analysis of Timed Systems*, ser. Lecture Notes in Computer Science. Springer International Publishing, 2014, vol. 8711, pp. 222–236.

[5] O. Maler and D. Nickovic, "Monitoring properties of analog and mixed-signal circuits," *STTT*, vol. 15, no. 3, pp. 247–268, 2013.

[6] M. M. Bersani, D. Bianculli, S. Dustdar, A. Gambi, C. Ghezzi, and S. Krstic, "Towards the formalization of properties of cloud-based elastic systems." in *Proc. of the Intl. Work. on Principles of Engineering Service-Oriented and Cloud Systems*, ser. PESOS '14, 2014, pp. 38–47.

[7] A. Gambi, W. Hummer, H. L. Truong, and S. Dustdar, "Testing elastic computing systems," *IEEE Internet Computing*, vol. 17, no. 6, pp. 76–82, Nov 2013.

[8] A. Gambi, A. Filieri, and S. Dustdar, "Iterative test suites refinement for elastic computing systems," in *Proc. of the Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE '13, 2013, pp. 635–638.

[9] A. Gambi, G. Toffetti, and M. Pezzé, *Assurance of Self-adaptive Controllers for the Cloud*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, vol. 7740, pp. 311–339.

[10] G. Galante and L. de Bona, "A survey on cloud computing elasticity," in *Proc. of the Intl. Conf. on Utility and Cloud Computing*, ser. UCC '12, Nov 2012, pp. 263–270.

[11] T. Lorido-Botran, J. Miguel-Alonso, and J. Lozano, "A review of auto-scaling techniques for elastic applications in cloud environments," *Journal of Grid Computing*, vol. 12, no. 4, pp. 559–592, 2014.

[12] S. Dustdar, Y. Guo, B. Satzger, and H. L. Truong, "Principles of elastic processes," *IEEE Internet Computing*, vol. 15, no. 5, pp. 66–71, Sept 2011.

[13] N. Herbst, S. Kounev, and R. Reussner, "Elasticity in cloud computing: What it is, and what it is not," in *Proc. of the Intl. Conf. on Autonomic Computing*, ser. ICAC '13, 2013, pp. 23–27.