Modeling and Evaluating Cross-layer Elasticity Strategies in Cloud Systems

Khaled Khebbeb, Nabil Hameurlain, Faiza Belala

Khaled Khebbeb, Nabil Hameurlain, Faiza Belala
"Modeling and evaluating cross-layer elasticity strategies in Cloud systems"
In 8th International Conference on Model and Data Engineering (MEDI 2018).
24 - 26 October 2018, Marrakesh, Morocco.
Published in: Lecture Notes in Computer Science (LNCS), vol 11163, pp. 168-183.
Publisher: Springer, Cham
DOI: https://doi.org/10.1007/978-3-030-32213-7_5

Modeling and Evaluating Cross-Layer Elasticity Strategies in Cloud Systems

Khaled KHEBBEB[1, 2], Nabil HAMEURLAIN[2], Faiza BELALA[1]

[1]LIRE Laboratory, Constantine 2 University – Abdelhamid Mehri, Constantine, Algeria
{khaled.khebbeb, faiza.belala}@univ-constantine2.dz
[2]LIUPPA Laboratory, University of Pau, Pau, France
{khaled.khebbeb, nabil.hameurlain}@univ-pau.fr

Abstract: Clouds are complex systems that provide computing resources in an elastic way. Elasticity property allows their adaptation to input workload by (de)provisioning resources as the demand rises and drops. However, due to the numerous overlapping factors that impact their elasticity and the unpredictable nature of the workload, providing accurate action plans to manage cloud systems' elastic adaptations is a particularly challenging task. In this paper, we propose an approach based on *Bigraphical Reactive Systems* (BRS) to model cloud structures and their elastic behavior. We design elasticity strategies that operate at service and infrastructure cloud levels to manage the elastic adaptations. Besides, we provide a Maude encoding to permit generic executability and formal verification of the elastic behaviors. One step ahead, we show how the strategies can be combined at both levels to provide different high-level elastic behaviors. Finally, we evaluate the different cross-layer combinations using *Queuing Theory*.

Keywords. Cloud Computing, Elasticity, Cross-Layer Elastic Behavior, Modeling, Bigraphical Reactive Systems, Maude.

1 Introduction

Cloud computing [25] is a recent paradigm that has known a great interest in both industrial and academic sectors. It consists of providing a pool of virtualized resources (servers, virtual machines, etc.) as on-demand services. These resources are offered by cloud providers according to three fundamental service models: infrastructure as a service (IaaS), platform as a service (PaaS), and software as a service (SaaS). The most appealing feature that distinguishes the cloud from other models is the elasticity property [16]. Elasticity [11] allows to efficiently control resources provisioning according to workload fluctuation in a way to maintain an adequate quality of service (QoS) while minimizing operating cost. Such a behavior is implemented by an elasticity controller: an entity usually based on a closed control loop [18] that decides of the elasticity actions to be triggered to adapt to the demand. In fact, managing a cloud system's elasticity can be particularly challenging. Elastic behaviors rely on many overlapping factors such as the available resources, current workload, etc. Managing these dependencies significantly increases the difficulty of modeling cloud systems' elasticity controller. To

address this challenge, formal methods characterized by their efficiency, reliability and precision, present an effective solution to deal with these numerous factors.

In this paper, we provide a formal modeling approach that reduces the complexity of designing cloud systems and the elasticity controller behavior. We adopt *Bigraphical Reactive Systems* (BRS) [26] as a *meta-model* for specifying structural and behavioral aspects of elastic cloud systems. Bigraphs are used to model the structure of cloud systems and the elasticity controller. Bigraphical reaction rules describe the elastic behavior of a cloud system. We focus on the infrastructure (IaaS) and service (SaaS) levels to define reactive elasticity strategies for provisioning and deprovisioning cloud resources in a cross-layered way. A strategy provides the logic that governs resources provisioning. It enables the elasticity controller to manage the cloud system's elastic behavior. It consists of a set of actions (bigraphical reaction rules) that are triggered according to the specified conditions (i.e., reactive strategies take the form: *if condition(s) then action(s)*).

Furthermore, we turn to Maude [23] as a semantic framework to encode the BRS modeling approach and to provide a generic executable solution of cloud elastic behavior. Maude is a formal tool environment based on rewriting logic. It can be used as a declarative and executable formal specification language, and as a formal verification system. It provides good representation and verification capabilities for a wide range of systems including models for concurrency. This enables us to easily map the BRS specifications into Maude modules and to manage the *non-determinism* that characterizes cloud systems' elastic behavior.

Finally, we present a way to combine different strategies at both infrastructure and service levels to enable different high level elastic behaviors. We propose a queuing-based approach as an analytical support for the elastic behavior. Precisely, we conduct experimental simulations of different execution scenarios to provide a quantitative evaluation of the multiple cross-layer elasticity strategies combinations.

The remainder of the paper is structured as follows. In Section 2, we present our vision of cloud systems and explain how their elastic behavior is managed by the elasticity controller. In Section 3, we introduce and use BRS formalism to provide a modeling approach for cloud systems. We model the elasticity controller and define elasticity strategies. In Section 4, we encode the bigraphical specifications of elastic cloud systems into Maude. We provide a quantitative evaluation of the elasticity strategies combinations using a queuing approach in Section 5. In Section 6, we review the state of art on elasticity and formal specification of elastic cloud systems. Finally, Section 7 summarizes and concludes the paper.

2 Cloud Systems and Elasticity

At a high level of abstraction, an elastic cloud system can be divided in three parts: the *front-end* part, the *back-end* part and the *elasticity controller*. The front-end represents the client interface that is used to access the cloud system and to interact with it. The back-end part refers to the cloud system's *hosting environment*, i.e., the set of computing resources (servers, virtual machines, service instances, etc.) that are deployed in the system and that are provided to satisfy its incoming workload. Cloud systems offer

their computing resources in an elastic way. *Elasticity* is property that was defined as "*the degree in which a system is able to adapt to workload changes by provisioning and deprovisioning resources in an autonomic manner such that at each point in time the available resources match the current demand as closely as possible.*" [13].

Elastic cloud systems usually work according to the closed-loop architecture shown in Figure 1, where the elastic cloud system receives *end-users*' requests through its client interface. The amount of received requests (i.e., the *input workload*) can oscillate in an unpredictable manner. The growing workload, thus the *system's load* can cause users *Quality of Experience (QoE)* degradations (e.g. performance drop). The *cloud infrastructure provider* hosts the controlled system (i.e., the cloud *hosting environment*). It provides costs to the cloud *service provider* according to the provisioned resources (that are allocated to the service provider's running applications). When the input workload drops, the eventually unnecessarily allocated resources are still billed. The *elasticity controller* monitors the controlled system and determines its adaptation (i.e., its *elastic behavior*). The adaptation actions (i.e., (de)provision cloud resources) are triggered to satisfy high-level policies that are set by the service provider such as *minimize costs, maximize performance*, etc.

The behavior of an elastic system can be intuitively described as follows. During its runtime, the system's load can increase. Which might lead to overload the provisioned resources. To avoid the saturation, an elastic system stretches, i.e., it scales by provisioning more computing resources. Conversely, when the system load decreases, some resources might become underused. To reduce costs, the elastic system contracts, i.e., it scales by deprovisioning the unnecessarily allocated resources [4]. However, due to the complexity of cloud systems and the multiplicity of the overlapping factors that impact their elasticity, specifying and implementing the elastic behavior is a particularly tedious task. Elasticity is specified by strategies that are designed to satisfy the high-level policies in an autonomic way. In this paper, we address this challenge by relying on formal methods. We provide a BRS based modeling of cloud systems' structure and the elasticity controller's behavior. Then we encode the proposed specification into Maude language to provide an executable solution of the elastic behaviors.

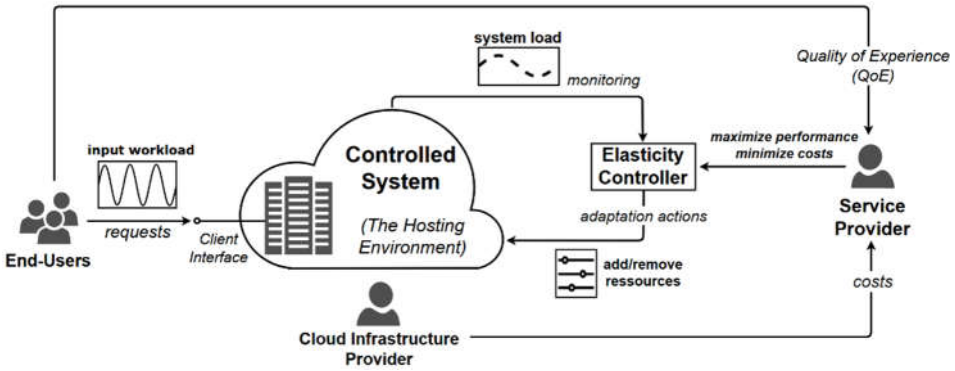

Fig. 1. High level view of cloud systems' elastic behavior

3 BRS Based Specification of Elastic Cloud Systems

Bigraphical reactive systems (BRS) are a recent formalism introduced by Milner [26, 27], for modeling the temporal and spatial evolution of computation. It provides an algebraical model that emphasize both *connectivity* and *locality* via a *link graph* and a *place graph* respectively. A BRS consists of a set of *bigraphs* and a set of *reaction rules*, which define the dynamic evolution of the system by specifying how the set of bigraphs can be reconfigured.

3.1 Bigraphical Modeling of Cloud Systems

A cloud system is represented by a bigraph *CS* including all cloud architectural elements. The sorting logic introduces mapping rules and expresses all the constraints and formation rules, that *CS* needs to satisfy, to ensure proper and accurate encoding of the cloud semantics into BRS concepts. Formal definitions are given in what follows.

Definition 1. Formally, a cloud system is defined by a bigraph CS, where:

$$CS = (V_{CS}, E_{CS}, ctrl_{CS}, CS^P, CS^L): I_{CS} \rightarrow J_{CS}$$

- V_{CS} *and* E_{CS} are sets of nodes and edges of the bigraph *CS*.
- $ctrl_{CS} : V_{CS} \rightarrow K_{CS}$ a control map that assigns each node $v \in V_{cs}$ with a control $k \in K_{cs}$.
- $CS^P = (V_{CS}, ctrl_{CS}, prnt_{CS}): m_{CS} \rightarrow n_{CS}$ is the place graph of *CS* where $prnt_{CS}: m_{CS} \uplus V_{CS} \rightarrow V_{CS} \uplus n_{CS}$ is a parent map. m_{CS} and n_{CS} are the number of sites and regions of the bigraph *CS*.
- $CS^L = (V_{CS}, E_{CS}, ctrl_{CS}, link_{CS}): X_{CS} \rightarrow Y_{CS}$ represents link graph of *CS*, where $link_{CS}: X_{CS} \uplus P_{CS} \rightarrow E_{CS} \uplus Y_{CS}$ is a link map, X_{CS} and Y_{CS} are respectively inner and outer names and P_{CS} is the set of ports of *CS*.
- $I_{CS} = < m_{CS}, X_{CS} >$ *and* $J_{CS} = < n_{CS}, Y_{CS} >$ are the inner and outer interfaces of the cloud system bigraph *CS*.

Nodes V_{CS} represent the physical (servers) or logical (VM and service instances) elements of the cloud system. Edges E_{CS} represent the links (e.g. communication canals) that connect the nodes via their ports P_{CS}. Control map $ctrl_{CS}$ associate semantics to the nodes. The place graph CS^P gives the hierarchical construction of the system basing on the parent map $prnt_{CS}$ for nodes and regions (e.g. a server node is a parent for a VM node, or hosts is). Regions represent the different parts of the system (e.g. the hosting environment). Sites are used to neglect parts of the system that are not included in the model. The link graph CS^L gives the link map $link_{CS}$ that show all the connections between ports and names. Inner and outer interfaces I_{CS} and J_{CS} give the openness of the system to its external environment (other bigraphs). Inner and outer names X_{CS} and Y_{CS} give labels to different parts of the system for interfacing purposes.

Definition 2. The sorting discipline associated to *CS* is a triple $\Sigma_{CS} = \{\Theta_{CS}, K_{CS}, \Phi_{CS}\}$.

Where Θ_{CS} is a non-empty set of sorts. K_{CS} is its signature, and Φ_{CS} is a set of formation rules associated to the bigraph. Table 1 gives for each cloud concept the

mapping rules for BRS equivalence. It consists of the control associated to the entity, its arity (number of ports) and its associated sort. *Sorts* are used to distinguish node types for structural constraints while *controls* identify states and parameters a node can have. For instance, a server noted SE has control SE^{L} when it is overloaded and SE^{U} when unused but all nodes representing servers are of sort e.

Table 1. The sorting discipline of the bigraph *CS*

Cloud element	Control	Arity	Sort
Server	SE	2	e
Overloaded server	SE^{L}	2	e
Unused server	SE^{U}	2	e
Virtual machine	VM	2	v
Overloaded VM	VM^{L}	2	v
Unused VM	VM^{U}	2	v
Service instance	S	1	s
Overloaded service instance	S^{L}	1	s
Unused service instance	S^{U}	1	s
Request	q	0	r

Table 2 gives the formation rules that define construction constraints over the bigraphical model. Rule Φ0 specifies that servers are at the top of the hierarchical order of the deployed entities in the bigraph. Rules Φ1-3 give the structural disposition of the hosting environment where a server hosts VMs, a VM runs service instances and a service instance handles requests. All connections are port-to-port links to illustrate possible links between the different cloud entities. In Φ5-6, we use the name *w* (for workload) to illustrate the connection the cloud system has with its abstracted front-end part. A server is linked to its hosted VMs and a VM is linked to the service instances it is running [19]. Rule Φ4 gives the active elements, i.e., that may take part in reactions.

Table 2. Construction constraints Φ_{CS} of the bigraph *CS*

	Rule description
Φ0	All children of a 0-region (hosting environment) have sort e
Φ1	All children of a e-node have sort v
Φ2	All children of a v-node have sort s
Φ3	All children of a s-node have sort q
Φ4	All $\overline{\mathrm{evsq}}$-nodes are active
Φ5	In an e-node, one port is always linked to a w-name and the other may be linked to v-nodes
Φ6	In a v-node, one port is always linked to a e-node and the other may be linked to s-nodes

3.2 The Elasticity Controller as a Behavioral Entity

The elasticity controller determines the adaptations of the cloud system's hosting environment. In our modeling approach, we consider this entity as the set of reaction rules that describe the system's behavior and the logic that governs the rules' triggering. This logic is implemented as *strategies* that describe different adaptations of the cloud system in a *cross-layered* manner (i.e., at infrastructure and service cloud levels).

Reaction Rules. A reaction rule Ri is a pair (R, R'), where *redex* R and *reactum* R' are bigraphs that have the same interface. The evolution of the cloud bigraph CS is derived by checking if R is a match in CS and by substituting it with R' to obtain a new system CS'. This is made with triggering the suitable reaction rule Ri. The evolution is noted $CS \xrightarrow{\text{Ri}} CS'$.

Table 3 gives the algebraic description of the different reaction rules that implement the adaptation actions of the elasticity controller. Sites (expressed as d) are used to neglect the elements that are not included in the reaction. The specified rules define the *horizontal scale* elasticity actions at different cloud levels. Reaction rules are applied for provisioning (R1-2) and deprovisioning (R3-4) resources by *scaling-out* and *scaling-in* the hosting environment. Rules R5-6 specify migration actions for service instances and requests, which are used to balance the system's load.

Table 3. Reaction rules describing adaptation actions

Adaptation action	Reaction rule algebraic form
	Scale-Out
Replicate service instance	$\text{R1} \stackrel{\text{def}}{=} \text{SE.}\big((\text{VM.}(\text{S.d2})\vert\text{d1})\big\vert\text{d0}\big)\vert\text{id} \rightarrow \text{SE.}\big((\text{VM.}(\text{S.d2})\vert\text{S})\big\vert\text{d1}\big)\vert\text{d0})\vert\text{id}$
Replicate VM instance	$\text{R2} \stackrel{\text{def}}{=} \text{SE.}\big((\text{VM.}(\text{S.d2})\vert\text{d1})\big\vert\text{d0}\big)\vert\text{id} \rightarrow \text{SE.}(((\text{VM.}(\text{S.d2})\vert\text{d1})\vert(\text{VM}))\vert\text{d0})\vert\text{id}$
	Scale-In
Consolidate service instance	$\text{R3} \stackrel{\text{def}}{=} \text{SE.}((\text{VM.}(\text{S.d3})\vert(\text{S.d2})\vert\text{d1})\vert\text{d0})\vert\text{id} \rightarrow \text{SE.}((\text{VM.}(\text{S.d2})\vert\text{d1})\vert\text{d0})\vert\text{id}$
Consolidate VM instance	$\text{R4} \stackrel{\text{def}}{=} \text{SE.}(((\text{VM.}(\text{S.d3})\vert\text{d2})\vert(\text{VM.d1}))\vert\text{d0})\vert\text{id} \rightarrow \text{SE.}((\text{VM.}(\text{S.d2})\vert\text{d1})\vert\text{d0})\vert\text{id}$
	Load Balancing
Migrate service instance	$\text{R5} \stackrel{\text{def}}{=} \text{SE.}(((\text{VM.}(\text{S.d3})\vert\text{d2})\vert(\text{VM.d1}))\vert\text{d0})\vert\text{id} \rightarrow \text{SE.}(((\text{VM}\vert\text{d2})\vert(\text{VM.}(\text{S.d3})))\vert\text{d0})\vert\text{id}$
Transfer request	$\text{R6} \stackrel{\text{def}}{=} \text{SE.}\big((\text{VM.}(\text{S.q}\vert\text{d4})\vert\text{d3})\vert(\text{VM.}(\text{S.d2})\vert\text{d1})\vert\text{d0}\big)\vert\text{id} \rightarrow \text{SE.}\big((\text{VM.}(\text{S.d4})\vert\text{d3})\vert(\text{VM.}(\text{S.q}\vert\text{d2})\vert\text{d1})\vert\text{d0}\big)\vert\text{id}$

Elasticity Strategies. As explained before, the specified strategies define the logic that governs the elastic behavior of the controlled cloud system. We use reactive strategies to make decisions about the elastic adaptations of the deployed entities by reasoning on their states. A reactive strategy takes the form: *IF Condition(s) THEN Action(s)* where conditions are expressed in *predicates logic* and actions are reaction rules. Table 4 defines the scaling (out/in) policies at both service and infrastructure levels.

Infrastructure Level. We introduce two strategies to express different provisioning policies for VM instances, as follows.

- *Strategy V1:* ensures VM instances' *high availability*. It states that the system *scales out*, i.e., provision a new VM instance, by executing rule R2 when at least one VM is overloaded, i.e., when it reaches its upper threshold of hosted service instances. In other terms, when it has control VM^{L}.
- *Strategy V2:* is designed to ensure the *limited availability* in terms of VM instances. It states that scale-out adaptations (provisioning VM instances) are triggered when all available VMs are overloaded.
- Both *V1* and *V2* specify that the system *scales-in*, i.e., deprovisions an empty VM instance (of control VM^{U}) by executing rule R4, if one is detected and no overloaded VM is available. This choice prevents having contradictory adaptation loops.

Service *Level.* We define two strategies to describe the system's service instances provisioning behaviors, as follows.

- *Strategy S1*: ensures service instances' *high availability*. It states that a new instance of service is provisioned by executing rule R1, when at least one available instance is overloaded (when it has control S^L).
- *Strategy S2:* defines service instances' *limited availability*. It states that *scale-out* adaptations (provisioning service instances) are triggered when all available service instances are overloaded.
- Strategies *S1* and *S2* specify that the system *scales in*, i.e., deprovisions an empty service instance (which has control S^U) by executing rule R3, when one is detected, and no overloaded instance is available.

Table 4. Scaling strategies at service and infrastructure levels

Strategy	Scale-Out	Scale-In
	Infrastructure level	
V1	$IF\ \exists v \in V_{CS}\ ctrl_{CS}(v) = \mathrm{VM}^{\mathrm{L}}\ THEN\ R2$	$IF\ \forall v \in V_{CS}\ \exists v' \in V_{CS}\ ctrl_{CS}(v) \neq \mathrm{VM}^{\mathrm{L}}$
V2	$IF\ \forall v \in V_{CS}\ ctrl_{CS}(v) = \mathrm{VM}^{\mathrm{L}}\ THEN\ R2$	$\wedge\ ctrl_{CS}(v') = \mathrm{VM}^{\mathrm{U}}\ THEN\ R4$
	Service level	
S1	$IF\ \exists s \in V_{CS}\ ctrl_{CS}(s) = \mathrm{S}^{\mathrm{L}}\ THEN\ R1$	$IF\ \forall s \in V_{CS}\ \exists s' \in V_{CS}\ ctrl_{CS}(s) \neq \mathrm{S}^{\mathrm{L}}$
S2	$IF\ \forall s \in V_{CS}\ ctrl_{CS}(s) = \mathrm{S}^{\mathrm{L}}\ THEN\ R1$	$\wedge\ ctrl_{CS}(s') = \mathrm{S}^{\mathrm{U}}\ THEN\ R3$

In addition, we define two strategies for the system's load balancing at both service and infrastructure levels as follows.

- *Strategy LB-V*: describes the system load balancing at infrastructure level, it states that service instances are migrated from loaded VMs to less loaded ones (executing rule R5) to reach a VMs load equilibrium.
- *Strategy LB-S*: states that requests are transferred from loaded service instances to less loaded ones (by applying rule R6) to achieve load balancing at service level.

Modeling the Elastic Behavior with LTL. Modeling the introduced elastic behavior with *Linear Temporal Logic* allows the specification of formulas to verify the system's elastic adaptations. To this purpose, we define a model of temporal logic with a *Kripke structure* $\mathbf{A_{CS}}$, as follows.

Definition 3. Given a set AP_{CS} of *atomic propositions*, we consider the Kripke structure $\mathbf{A_{CS}} = (A, \rightarrow_{\mathbf{A}}, L_{CS})$. Where A is the *set of states*, $\rightarrow_{\mathbf{A}}$ is the *transition relation*, and $L_{CS}: A \rightarrow AP_{CS}$ is the *labeling function* associating to each state $a \in A$, the set $L_{CS}(a)$ of the atomic propositions in AP_{CS} that hold in the state a. $LTL(AP_{CS})$ denotes the formulas of the *propositional linear temporal logic*. The semantics of $LTL(AP_{CS})$ is defined by a *satisfaction relation*: $\mathbf{A_{CS}}, a \vDash \varphi$, where $\varphi \in LTL(AP_{CS})$.

We consider the set $AP_{CS} = \{Stable, Overloaded, Underused, LBTrue, \mathrm{M}\}$ of the atomic propositions that describe the hosting environment's states. For the sake of simplicity, these states are symbolic and relate to the elastic behavior of the system. The system is considered *Overloaded*/*Underused* when at least one entity (VM, Service) is overloaded/unused. It is *Stable* otherwise. *LBTrue* is a non-exclusive proposition that

can hold together with *Stable, Overloaded* or *Underused* (that are exclusive) when load balancing at VM or Service levels is applicable. M holds when the system is being monitored. In other terms, different structural states of the system in A (i.e., configurations) can be gathered (i.e., labeled) in the same class of equivalence with respect to the global symbolic state of the system in AP_{CS}.

The *non-deterministic finite-state automaton* in Figure 2 shows the transitions *Scale-Out*, *Scale-In* and *LB* (for *Load Balancing*) that represent the adaptation actions that are executed by the elasticity controller. The transitions *Input* and *Output* stand for receiving and releasing end-users' requests. Initially, the controlled system is in the monitoring phase. When monitored, it can be at any elastic state.

Note that the evolution of the system's state depends on its elastic constraints (bounded resources capacity introduced by thresholds, triggering predicates, etc.). Thus, reaching the stable state is not always possible (i.e., all elastic states can be final).

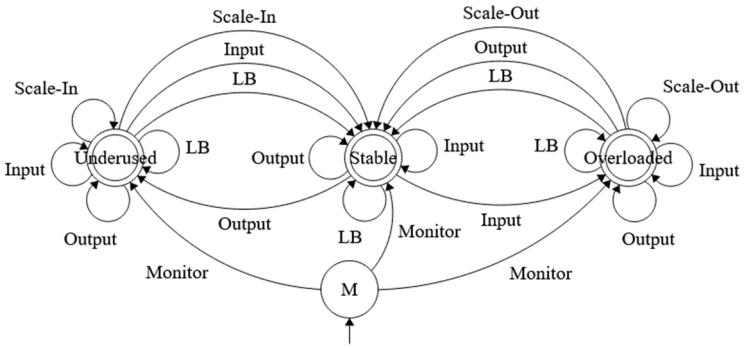

Fig. 2. Elastic behavior non-deterministic finite-state automaton

To describe the elastic behaviors that are triggered by the elasticity controller in LTL, we introduce the set $LTL(AP) = \{Scale\text{-}Out, Scale\text{-}In, LoadBalance\}$ of the propositional formulas, as follows.

- $Scale\text{-}Out \equiv \boldsymbol{G}\,(\,Overloaded \rightarrow \boldsymbol{F}\,Stable\,)$
- $Scale\text{-}In \equiv \boldsymbol{G}\,(\,Underused \rightarrow \boldsymbol{F}\,Stable\,)$
- $LoadBalance \equiv \boldsymbol{G}\,(\,LBTrue \rightarrow \boldsymbol{F} \sim LBTrue\,)$

Where the formulas *Scale-Out* and *Scale-In* state that a given system that is *Overloaded*/*Underused* will eventually reach its *Stable* state. *LoadBalance* formula ensures that the system will eventually apply load balancing as long as it is possible. We use the symbol ~ for negation. The symbols $\boldsymbol{G}$ and $\boldsymbol{F}$ are LTL operators that respectively stand for "*always*" and "*eventually*".

4 Principles of Maude Encoding and Property Verification

To verify the correctness of the introduced elasticity strategies and to watch the aimed cross-layered elasticity, it is important to provide an executable solution for the

specified elastic behaviors. Theoretically, BRS provide good *meta-modeling* bases to specify cloud systems' structure and their elastic behavior. As for their executable capabilities, the few existing tools built around BRS as BigraphER [5] and BPL Tool [14] are limited and only suitable for some specific application domains. Furthermore, the BRS *model-checker* BigMC [30] that was used in [32], allows formal verification of safety properties. However, the possible verifications rely on very limited predefined predicates. These tools lack of providing concurrent and autonomic executability of the specified BRS models. In this paper, we turn to Maude language to tackle these limitations and to provide a generic executable solution of elasticity strategies together with their verification.

4.1 Motivating the Use of Maude

Maude [9] is a high-level formal specification language based on equational and rewriting logics. A Maude program is a *logical theory* and a Maude computation is *logical deduction* which uses the axioms specified in the program/theory. A Maude specification is structured in two parts. (1) A functional module that specifies a *theory* in membership equational logic. Such a theory is a pair $(\Sigma, E \cup A)$, where the *signature* Σ specifies the type structure (sorts, subsorts, operators etc.). E is the collection of the (possibly conditional) equations declared in the functional module, and A is the collection of equational attributes (associative, commutative, etc.) declared for the operators. (2) And a system module that specifies a *rewrite theory* as a triple $(\Sigma, E \cup A, R)$. Where $(\Sigma, E \cup A)$ is the module's equational theory part, and R is a collection of (possibly conditional) rewrite rules.

The Bigraphical specifications for cloud systems' structure (in Section 3.1) can be encoded in a functional module. Where the declared operations and equations define the constructors that build the system's elements. Similarly, BRS dynamics (in Section 3.2) that describe the elasticity controller's behavior can be encoded in a system module. Where the elasticity strategies are described as conditional rewrite rules. The set of rewrite rules R express the bigraphical reaction rules. Their triggering conditions expressed as equations from the functional module encode the strategies' predicates.

4.2 Setting Up Elastic Cloud Systems

To encode the BRS modeling approach for cloud structures and their elastic behavior in Maude, we first map the BRS model into Maude language as shown in Table 5.

Structure Encoding. In the functional module, the bigraph sorts e, v and s (i.e., server, VM and service) are defined as CS, VM and S. Note that we enriched Maude sorts with additional information as the maximum hosting thresholds and the entities states. A sort is built according to its associated constructor. For instance, a cloud server is built by the term `CS<x,y,z/VML:state>`, where `x`, `y` and `z` are naturals that encode upper hosting thresholds at server, VM and service levels. `VML` is a list of VMs, this relationship is expressed by the declaration of sort VM as a subsort of sort VML. The element `state` gives a state out of the constructors (overloaded, unused, stable, etc.). To enable horizontal scale strategies according to configurable preferences, we define the sort

`HSCALE(V i, S j) :: cs`. Where the parameters $i, j \in [1,2]$ indicate which strategies are applied at infrastructure (*V1* or *V2*) and service (*S1* or *S2*) levels of the cloud system `cs`.

Table 5. Encoding the BRS cloud model into Maude

Bigraphical model	Maude specification
	Functional module

Sorting discipline (*structure construction*)	`sorts HSCALE CS VM S VML SL state . subsort VM < VML . subsort S < SL .`
	`op HSCALE (V_ , S_) :: _ : Nat Nat CS -> HSCALE [ctor] .`
	`op CS<_,_,_/_:_> : Nat Nat Nat VML state -> CS [ctor] .`
	`op VM{_,_:_} : Nat SL state -> VM [ctor] .`
	`op S[_,_:_] : Nat Nat state -> S [ctor] .`
	`ops stable over under idle : -> state [ctor] .`
	`op nilv : -> VML [ctor] . op _|_ : VML VML -> VML [ctor assoc comm id: nilv] .`
	`op nils : -> SL [ctor] . op _+_ : SL SL -> SL [ctor assoc comm id: nils] .`
	…

System state predicates (*self-aware property*)	`ops isStable(_) isOverloaded(_) isUnderused(_) AoverV(_) EoverV(_) EunV(_)`
	`AoverS(_) EoverS(_) EunS(_) LBVpred(_) LBSpred(_) : CS -> Bool .`
	…

	System module
Reaction rules and elasticity strategies (*self-adaptive property*)	Conditional rewrite rules of the form: `crl [rewrite-rule-name] : term => term' if condition(s) .`

System State Predicates Encoding. We define a set of system predicates in the functional module that give information about the managed cloud system configuration (that we express as a cloud server in Maude). For instance, *AoverV()* is a predicate for "*all VMs are overloaded*" and *EunS()* is a predicate for "*there exists an unused service instance*". We also encode system state predicates *isStable()*, *isOverloaded()* and *isUnderused()* that are true if the cloud system is stable, overloaded or underused.

Elasticity Strategies Encoding. Strategies are encoded as conditional rewrite rules in the system module. Their conditions are the states and monitoring predicates and their actions (bigraph reaction rules) are encoded as Maude functional computation. For instance, load-balancing strategy at VM level is specified as the following rewrite rule: `crl[LB-VM-level]:cs => LBV(cs) if LBVpred(cs).` Where `cs` is a given cloud system, `LBV(cs)` is an equation that reduces the term `cs` in such a way to apply load-balancing at VM level and `LBVpred(cs)` is a predicate that is true if load-balancing at VM level in `cs` is possible. *LBV()* and *LBVpred()* are defined as equations in the functional module.

Formal Verification of Elasticity. To verify the elastic behavior of the system as encoded in the system module, we define a Maude property specification based on *Linear Temporal Logic*. Maude allows associating *Kripke* structures to the rewrite theory specified in the system module. The semantics introduced by the Kripke structure $\mathbf{A}_{CS}$ in Section 3.2 allowed us to define a generic LTL model checking that can reason on any system configuration. For instance, determining that a cloud configuration is stable in terms of elasticity is specified with: `cs ⊨ Stable = true if isStable(cs) == true`. Where `cs` is a given cloud configuration. `Stable` is a proposition $\in AP_{CS}$ that represent the symbolic elastic state "*stable*". And `isStable(cs)` is a predicate for "*the cloud system cs is stable*" which is defined in the functional module.

We execute Maude's LTL model-checker with, as parameters, a cloud configuration as an initial state and a property formula in $LTL(AP_{CS})$ to verify. The model-checker can give counter examples showing the succession of the triggered rewrite rules that are applied on the initial state of the system, in such a way to verify the given property according to the specified elasticity strategies.

5 A Queuing Approach for Quantitative Evaluation

As its input workload rises, the congestions that may result in a system are in fact waiting queues that indicate the insufficiency of the provisioned resources. For this reason, we advocate that a queuing approach is a relevant support to study the elastic behavior of a system and to evaluate the performance of elasticity strategies. To proceed to a quantitative evaluation of the introduced strategies, we perform queuing-based offline simulations of elastic cloud systems.

Queuing Model. We consider a *queuing model*, defined by a set of parameters as introduced by the *Kendall* notation: A/S/C/Q/N/D [3], where C is the number of service instances. A is the *arriving process* describing how the requests arrive into the system. D is the *serving discipline* describing how the requests are processed (e.g., *first come first served*). The *service process* S gives the amount of time required to process the requests. Q is the maximum number of requests that the system can hold, and N is the number of requests expected to arrive into the system. In our evaluation, we consider that Q and $N = \infty$. We consider that A is a *Poisson process* which gives an exponential distribution of the received requests (at each time unit) with the average value of λ. S also follows an exponential law with the average value of μ to give the number of requests that are processed by service instances. The essence of elasticity being the adaptations, we use a queuing model with on-demand number C of service instances, inspired from [15], to show how the system adapts to its varying input workload by (de)provisioning resources at service and infrastructure levels.

Experiment. To evaluate elasticity, we consider the example of a cloud-based voting service where initially one VM is provisioned in which one service instance is deployed. We define the upper-bound hosting thresholds $v = 2$, $s = 2$ and $w = 40$, for the cloud system, the VMs and the service instances respectively in terms of VMs, service instances and requests. We simulate the execution of a cloud system from the same initial configuration according to the defined strategies, for both infrastructure (*V1, V2*) and service (*S1, S2*) levels. The simulations are performed within 50 time units over a scenario where $\lambda = 50$ and $\mu = 35$. The results give the system's average *resources provisioning*, *performance* and *efficiency*. Introducing thresholds makes the systems bounded in terms of hosting capabilities. Thus, the displayed rates are given in function of the maximum capacity of service/VM instances and their average recorded deployment. Idem for the system load (i.e., the processed requests per time unit). The delay represents the ratio between the pending requests and those being served.

Knowing that load balancing (*LB-V, LB-S*) is applied when possible, the graphs in Figure 3 show the cross-layer behaviors resulting from combining the scaling strategies introduced in Section 3.2.

Intuitively, combining high availability for both infrastructure and service levels (*V1, S1*) leads to *high-performance*, i.e., low processing delay (1%), but also brings high provisioning costs, i.e., high hosting environment deployment (93% service and 100% VM instances capacity).

Inversely, applying limited availability at both levels (*V2, S2*) implies low costs i.e., *high economy* but also low performance, i.e., high processing delay (28%).

The combination (*V2, S1*) ensures *infrastructure costs optimization*, i.e., new VMs are provisioned only when the available ones are fully loaded (by scaling-out at service level). It brings better overall optimization than (*V1, S1*) with less average service deployment and better average system load (with respectively 73% and 44% service instances provisioning and usage rate for combination (*VS, S1*) versus 93% and 32% for combination (*V1, S1*)), yet with lower performance (i.e., higher delay).

The combination (*V1, S2*) doesn't seem to describe a specific behavior (labeled "x") in this simulation. It leads to mediocre rentability of the VMs and to consequent delay regarding the recorded usage rate of the Service instances.

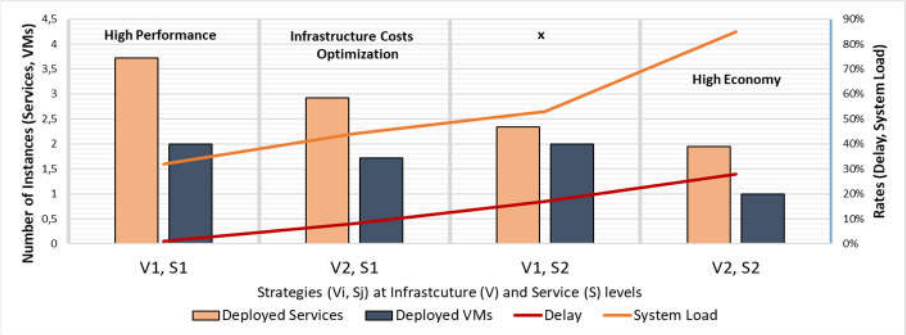

Fig. 3. Evaluation of cross-layer elasticity strategies

To conclude this evaluation, we want to emphasize the fact that the concept of "good" strategy is not absolute. It depends on the case study (i.e., the system configuration, workload tendencies, available resources, etc.) and on the preferences set by the cloud service provider [29]. Indeed, having strategies that describe different high-level behaviors gives a certain range of possibilities to endow the managed cloud system with the desirable elastic behaviors.

6 Related Work

There have been several researches in the literature about cloud systems' elasticity such as [21, 7, 1, 10, 33]. However, only a few works like [12, 20, 22, 28] were proposed to study elasticity property using formal methods.

In the context of modeling cloud systems and their elastic behaviors, authors in [4] adopted the temporal logic named CLTLt(D) (Timed Constraint LTL) to model some properties related to cloud systems such as elasticity, resource management and quality of service. In their work, they considered cloud resources as virtual machines and did not address service level. In [2] authors proposed a Petri Nets based formalization to

describe cloud-based business processes' elastic behaviors. They introduced elasticity strategies for routing, duplicating and consolidating cloud components at service level. They focused on the application layer of a cloud configuration but did not address the cloud infrastructure in their model. As for our adopted formalism, BRS were proven useful in the specification of ubiquitous, context aware and distributed systems [24, 17] and in other domains [6]. BRS were used in [31] to provide a generic model of elastic cloud systems. Authors modeled cloud structures with bigraphs in three parts: the front-end part, the back-end part and the elasticity controller. They relied on bigraphical reaction rules to express the front/back-end interactions along with the adaptation actions of cloud configurations at service and infrastructure levels. However, they lacked providing elasticity strategies that operate in an autonomic manner.

In our previous work [19], we proposed a BRS modeling for elastic cloud systems in two parts. First, we defined a bigraphical specification for the hosting environment and the elasticity controller structures. And second, we used bigraphical reaction rules to model the adaptation actions, which describe the elasticity controller's behavior.

In this present paper, we propose a different approach. We use a bigraphical modeling to describe the structural aspect of a cloud system's hosting environment *only*; and we model the elasticity controller as a behavioral entity. The controller is modeled using bigraphical reaction rules alongside with the logic that triggers the reactions. This logic is represented by elasticity strategies that specify the elastic behavior of the cloud system in a *cross-layered* manner (i.e., at service and infrastructure levels). This new approach enables seeing the elasticity controller as an intrinsic entity of the cloud system. Therefore, monitoring tasks over the controlled cloud system enables considering it as "*self-aware*"; and the adaptation actions that are triggered in function of its state enables considering it as "*self-adaptive*" [8]. In addition, we propose a way to combine the different designed strategies to provide multiple cross-layer elastic patterns. We evaluate the combinations to highlight the resulting high-level elastic behaviors.

Besides, *Control Theory* was used for resources management in distributed [35] and cloud [34] systems. One of the main limitations of this approach is the non-linearity of most inter-relationships in computing systems [36]. This requires designing nonlinear and adaptive controllers that are difficult to understand and implement. In this paper, we inspire from closed-loop based approaches to design our elasticity controller. It aims at having the controlled cloud system reach a "stable" global state (which is defined in predicates logic) by relying on elasticity strategies we specified using BRS. Maude encoding of these behaviors ensures autonomic and concurrent execution of the elastic adaptations. And Maude's LTL model-checking enables verifying the correctness of the adaptations regarding the reachability of the "stable" state.

7 Conclusion

In this paper, we provided a modeling approach for cloud systems' structure and elastic behaviors based on Bigraphical Reactive Systems. We use bigraphs and bigraphical reactive rules to express both aspects respectively. These behaviors implement the elasticity controller and are described by elasticity strategies. We propose different strategies for horizontal scale (de)provisioning of cloud system resources and for load

balancing at service and infrastructure levels. Strategies describe the logic that enables the elasticity controller to reason over the entire cloud system's state and manage its elastic adaptations.

One step further, we encoded the modeling approach into Maude language to provide a generic executable solution for elasticity in cloud systems. We also provided formal verification of elasticity property using the LTL model-checker integrated in Maude.

Besides, we presented an original way to compose different elasticity strategies at both service and infrastructure levels to provide multiple high-level elastic behaviors.

Finally, we proposed a queuing-based approach to conduct experimental simulations of the different elasticity strategies combinations in order to provide a quantitative evaluation of the adaptations.

As on-going work, we aim to enlarge the specifications of cloud system's elastic behavior. Our goal is to provide a more complete solution that considers vertical scale elasticity for cloud resources management.

8 References

[1] A. Ali-Eldin, J. Tordsson, and E. Elmroth, "An adaptive hybrid elasticity controller for cloud infrastructures", 2012 IEEE Network Operations and Management Symposium, Maui, HI, (2012), pp. 204–212.

[2] M. Amziani. "Modeling, evaluation and provisioning of elastic service-based business processes in the cloud". Thesis. Institut National des Télécommunications, 2015. English. <NNT: 2015TELE0016>. <tel-01217186>.

[3] B. Baynat, "Théorie des files d'attente". Paris: Hermès Science publications, 2000. Available online : http://books.google.fr/books?id=NWWgMQEACAAJ

[4] M. Bersani, D. Bianculli et al., "Towards the formalization of properties of cloud based elastic systems", Proceedings of the 6th International Workshop on Principles of Engineering Service-oriented and Cloud Systems – PESOS 2014, Hyderabad, 2014, pp. 38–47.

[5] M. Sevegnani and M. Calder, "BigraphER: rewriting and analysis engine for bigraphs". In proceedings of Computer Aided Verification (CAV 2016), Lecture Notes in Computer Science, Volume 9780 Part II, pp 494-501, Springer, Toronto, Canada, July 2016.

[6] M. Calder and M. Sevegnani, "Modeling IEEE 802.11 CSMA/CA RTS/CTS with stochastic bigraphs with sharing". Formal Aspects of Computing. 26(3): 537-561 (2014)

[7] K. Chatziprimou, K. Lano and S. Zschaler, "Runtime Infrastructure Optimization in Cloud IaaS Structures". CloudCom (1) (2013): 687-692

[8] T. Chen, R. Bahsoon and X. Yao, "A Survey and Taxonomy of Self-Aware and Self Adaptive Cloud Autoscaling Systems", ACM Computing Surveys, Vol. 1, No. 1, Article 1. January 2018.

[9] M. Clavel, F. Duran and al., "Maude Manual V 2.7.1". 2017

[10] G. Copil, D. Moldovan et al., "Multi-level elasticity control of cloud services", Service-oriented Computing, 2013, pp. 429–436.

[11] S. Dustdar, Y. Guo, B. Satzger, and H. Truong, "Principles of elastic processes", *IEEE Internet Comput.* 15 (2011), pp. 66–71.

[12] L. Freitas and P. Watson, "Formalizing workflows partitioning over federated clouds: Multi-level security and costs". *International Journal of Computer Mathematics*, 91(5), pp.881-906.

[13] G. Galante and L. Bona, "A survey on cloud computing elasticity", 2012 IEEE Fifth International Conference on utility and Cloud Computing, Chicago, Il, 2012, pp. 263–270.

[14] AJ. Glenstrup, TC. Damgaard et al., "An implementation of bigraph matching". Technical Report 2010-135. Copenhagen: ITUniversitetet Kobenhavn; 2010.

[15] A. Gurtov and V. Mazalov, "Queueing System with On-Demand Number of Servers", Mathematica Applicanda, vol. 40, no. 2, 2012.
[16] N. Herbst, S. Kounev, and R. Reussner, "Elasticity in cloud computing: What it is, and what it is not", In Proceedings *of the 10th International Conference on Autonomic Computing*, San Jose, CA: uSENIX, 2013.
[17] J. Wang, D. Xu and Z. Lei, "Formalizing the Structure and Behaviour of Context-aware Systems in Bigraphs". In First ACIS International Symposium on Software and Network Engineering; 2011.
[18] B. Jacob, "A Practical Guide to the IBM Autonomic Computing Toolkit". *IBM, International Technical Support Organization,* Raleigh, NC, 2004.
[19] K. Khebbeb, H. Sahli, N. Hameurlain et al., "A BRS Based Approach for Modeling Elastic Cloud Systems". Service-Oriented Computing – ICSOC 2017 Workshops, pp.5-17.
[20] S. Kikuchi and K. Hiraishi, "Improving reliability in management of cloud computing infrastructure by formal methods". In Network Operations and Management Symposium (NOMS); 2014. p. 1-7.
[21] L. Letondeur, "Planification pour la gestion autonomique de l'élasticité d'applications dans le cloud". Computer Science [cs]. Thesis at Joseph Fourier University, (2014). French. <tel-01140128>
[22] M. Rady, "Formal definition of service availability in cloud computing using OWL". In Computer Aided Systems Theory-EUROCAST. Springer; 2013. p. 189-194. N
[23] M. Clavel, F. Duran et al., "All about Maude. A High- Performance Logical Framework". volume 4350 of Lecture Notes in Computer Science. Springer, 2007.
[24] A. Mansutti, M. Miculan and M. Peressotti, "Multi-agent Systems Design and Prototyping with Bigraphical Reactive Systems". DAIS 2014: 201-208.
[25] P. Mell and T. grance, "The NIST Definition of Cloud Computing", *National Institute of Standards & Technology, Special Publication*, 2011, pp. 800–145.
[26] R. Milner, "Bigraphs and their algebra", Electron. Notes Theor. Comput. Sci. 209 (2008), pp. 5–19.
[27] R. Milner, "The Space and Motion of Communicating Agents", Cambridge university Press, Cambridge, 2009.
[28] A. Naskos, E. Stachtiari et al., "Cloud elasticity using probabilistic model checking" CoRR, vol. abs/1405.4699; 2014
[29] M. Netto, C. Cardonha et al., "Evaluating Auto-scaling Strategies for Cloud Computing Environments", 2014 IEEE 22nd International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems, 2014.
[30] G. Perrone, S. Debois, and T. Hildebrandt, "A Model Checker for Bigraphs". In proceedings of the 27th ACM Symposium in Applied Computing ACM-SAC'12; 2012.
[31] H. Sahli, N. Hameurlain and F. Belala, "A bigraphical model for specifying elastic cloud systems and their behaviour", International Journal of Parallel, Emergent and Distributed Systems, 2016 DOI: 10.1080/17445760.2016.1188927.
[32] H. Sahli, F. Belala and C. Bouanaka, "Model-Checking Cloud Systems Using BigMC" 8th International Workshop on Verification and Evaluation of Computer and Communication Systems. Bejaïa, Algeria. Sep 2014.
[33] D. Trihinas, C. Sofokleous et al., "Managing and monitoring elastic cloud applications", Lecture Notes in Computer Science, Toulouse, 2014, pp. 523–527.
[34] M. Mendieta, C. Martin et al., "A control theory approach for managing cloud computing resources: A proof-of-concept on memory partitioning". IEEE Second Ecuador Technical Chapters Meeting (ETCM). 2017.
[35] X. Liu, X. Zhu et al., "Adaptive entitlement control of resource containers on shared servers". 9th IFIP/IEEE International Symposium on Integrated Network Management, 2005.
[36] X. Zhu, M. Uysal et al., "What does control theory bring to systems research?". ACM SIGOPS Oper. Sys. Rev., vol. 43, no. 1, 2009.