A quality-driven approach for analyzing elastic cloud computing

Chafia Bouanaka, Esma Maatougui
LIRE Laboratory,
University of Constantine 2-Abdelhamid Mehri
Constantine, algeria
{chafia.bouanaka, esma.maatougui}@univ-constantine2.dz

Faiza Belala, Nadia Zeghib
LIRE Laboratory,
University of Constantine 2-Abdelhamid Mehri,
Constantine, Algeria
{faiza.belala, nadia.zeghib}@univ-constantine2.dz

Abstract

Cloud computing has emerged as a new computing paradigm that aims to provide on demand IT services with a rational and efficient use of resources but still maintaining the required Quality of Service(QoS) via elasticity key feature. However, resource usage patterns in elastic cloud are inherently probabilistic and provoke non-determinism while selecting the elasticity policy to be applied. The main objective of the present work is to supply solutions for the challenging task of managing elastic Cloud architecture by fully quantifying the non-determinism on the basis of QoS parameters. Hence, we define a formal approach that offers a model for specifying cloud architecture and its dynamics in terms of quality driven elasticity policies according to a continuous analysis of QoS parameters changes.

Keywords : Cloud Computing; Elasticity Policies; Formal Methods; PSMaude; Quantitative Verification.

1 Introduction

Cloud Computing is actually a major evolution of IT technology since it rationalizes computing resources assets at a worldwide scale, allowing companies to be more efficient while managing both the development and deployment costs of software systems. The most attractive feature of cloud systems is their ability to dynamically scale resources up or down over fine-grained time intervals [Joh 11], according to resource request variations over time. It also allows multiple users to be served simultaneously [Li 16].

Although the required resources of a cloud service are limited and statically determined, the workload can vary over time, and sometimes in an unpredictable way. This generally leads to an overloaded or unloaded cloud infrastructure causing fatal consequences on the quality of service level and the cloud service delivery cost. In this context, the elasticity management mechanism is faced with multiple obstacles for the growth and adoption of an efficient model for cloud systems. One major issue is the non-deterministic choice of the adequate elasticity strategy to be applied since a quick and dynamic resizing with respect to quality of service (QoS) parameters of the cloud architecture is not always evident. Additionally, resource usage patterns in elastic cloud systems are inherently probabilistic in nature and involve potentially unknown or non-deterministic factors [Joh 11]. Non-determinism is due to the various forms of elasticity (replication, consolidation and VM-migration) in the cloud model as well as the availability of several resources of the same nature whereas a dynamic reconfiguration is necessary to assure the elasticity property. Consequently, the elasticity property is quite complex and difficult to model, test and verify. Adopting a quantitative analysis approach to resolve the non-determinism; encountered while selecting both the elasticity mechanism to be adopted and the resource to be considered, is well suited.

The main objective of the present work is to supply solutions for the challenging task of managing elastic cloud systems with respect to QoS properties using formal methods and quantitative analysis. We propose a design methodology together with a formal model enabling the specification of the cloud architecture and its dynamics in terms of quality driven elasticity strategies by scaling up/down according to a continuous analysis of QoS parameters changes.

The paper is organized as follows: Section 2 presents the Ticket Booking system to be used to illustrate the various concepts introduced in the present work. Section 3 defines the PSMaude semantics to the proposed approach. In section 4, the model is validated through the verification of some cloud QoS properties. Section 5 rounds up the paper.

In: Proceedings of the 3rd Edition of the International Conference on Advanced Aspects of Software Engineering (ICAASE18), Constantine, Algeria, 1,2-December-2018, published at http://ceur-ws.org.

2 Motivating Example

The various concepts introduced in the present work are illustrated via a ticket booking system (Ticket Booking), inspired and adapted from [Ron 09]. The system is deployed in a cloud infrastructure and is composed of two services: Air Ticket Booking service (*S1*) for booking air tickets, and Boat Ticket Booking service (*S2*) for Boat Tickets Booking. Supposing that actually there are four users (*A1*, *A2*, *A3* and *B*) exploiting the Ticket Booking system, two of them (*A1* and *B*) are booking an Air ticket by requesting service (*S1*), and user *A2* is actually booking a boat ticket via service (*S2*). The two services are deployed on the same physical server (Server1) but running on two different virtual machines, *VM1* and *VM2* respectively. The second (*Server2*) and third (*Server3*) servers actually contain *VM3* and *VM4* virtual machines respectively. User requests are dispatched by the load balancer element of a cloud system.

Table 1 Correspondence between cloud architecture concepts and Maude

	Cloud concept	*Maude concept*
Cloud architecture	User	`class User | connected : Bool .`
	Service	`class Service | NbrClient : Nat, Type : Qid ,clients : OidListe, Cout : Nat .`
	Virtual Machine	`class Vm | NbrService : Nat , NbrReq : Nat , services : OidListe, state : State , Cout : Nat .`
	Server	`class Server | NbrVm : Nat , NbrReq : Nat , vms : OidListe, state : State , Cout : Nat .`
	Load balancer	`class LoadBalancer | connected : Bool .`
	Data center	`class DataCenter | loadbalancer : Oid , servers : OidListe .`
Elastic Cloud Dynamics	Elasticity Strategy	**`Probabilistic Rewrite rule`**
	Elasticity Strategy selector	**`PSMaude strategy`**

Throughout our motivating example, we attempt to identify major issues to be encountered while designing efficient elastic cloud systems that are constrained to adapt to workload variations in order to ensure certain QoS properties.

The principle of the elasticity property is to ensure the provisioning of necessary and sufficient resources such that a cloud service continues running smoothly even as the workload scales up or down, thereby avoiding under-utilization and overutilization of resources [Geel 09]. The elasticity property can be provided using three fundamental mechanisms: Service Replication, Service Consolidation and Service Migration. Accordingly, to face a workload variation, two or more elasticity strategies may be candidate at the same time. As an example, we suppose that the *VM1*virtual machine is overloaded and is no more able to treat requests for Air Ticket service *S1*, is it preferable to replicate it and thus create a new instance to treat user requests or redeploy (migrate) the requested service on a less loaded virtual machine? Consequently, the non-determinism arises while selecting the elasticity strategy to be applied. Besides, in certain situations, a given elasticity strategy is more reliable than others with respect to service delivery QoS parameters. Hence, we notice that elastic cloud systems exhibit both probabilistic and nondeterministic behaviors.

To address the above challenges, we attempt to quantify the non-determinism to obtain a fully probabilistic model by associating a cumulated cost attribute to each cloud resource. Then, a dynamic weighting, which is inversely proportional to the resource cost, of the elasticity strategies is performed to select the more adequate one. The proposed approach enables specifying and analyzing elastic cloud systems

with respect to QoS requirements that quantify the non-determinism.

3 Elastic Cloud Semantics

We associate a formal operational semantics to elastic cloud system. In particular, the aim of providing semantics is to specify the behavior of an elastic cloud system in terms of changes on its structure via the elasticity strategies and obtain an executable specification that can be then analyzed with respect to QoS parameters. We choose Maude [Cla 08] and its extension PSMaude[Ben 13] as the basis for the definition of elastic cloud semantics.
Maude is a high-performance language and system supporting both equational and rewriting logic [Mes 93] specification and programming for a wide range of systems and applications. Equational theories describe the static parts of a system and are represented as functional modules. Rewrite theories describe the dynamic parts of the system and are represented in Maude as system modules.

PSMaude [Ben 13] extends Maude by adding the necessary support for specifying both probabilistic rewrite rules and probabilistic strategies. It also provides a set of probabilistic rewrite commands together with a statistical PCTL model checker to analyze a given probabilistic rewrite theory being controlled by probabilistic strategies. In particular, PSMaude allows simulating and comparing the evolution of a "base" unquantified model under different probabilistic strategies, from a given initial state.

The proposed formalization approach is based on a set of formal mapping rules (see Table 1) defining the correspondences between elastic cloud system concepts and PSMaude ones. Structural aspects of a cloud system are mapped to a judiciously defined set of classes, elasticity strategies to probabilistic rewrite rules and the elasticity selector mechanism to a set of PSMaude strategies.

3.1 Cloud Architecture Formalization

To reflect the hierarchical structure (see table 1) of cloud systems and avoid structure flatting through algebraic terms, we adopt an object oriented approach in Maude. Thus, the cloud architecture is considered as a collection of objects that conform to a well defined structural hierarchy. Identifying cloud system elements as objects; each one with its own context, properties and actions, facilitates considerably the design and understandability of the model.

3.2 Elasticity Strategies Formalization

The dynamic aspect of the cloud architecture is specified via a set of rewrite rules expressing local changes on the cloud architecture in terms of resource elasticity while preserving the initial architectural constraints. Since we are interested with the provision of resources and the elasticity of the cloud architecture, the proposed rewrite rules allow allocating and releasing services or resources in general.

Three forms of elasticity can appear in Cloud architectures.

- *Service Replication:* Whenever the workload scales up while the deployed services are unable to treat all requests, the replication or duplication of services/VMs is necessary.
- *Service Consolidation:* Resource consolidation is the dual operation of the replication one. It is performed whenever the workload scales down and consists of deleting useless copies of services/VMs.
- *Service/VM Migration:* This situation appears when a server becomes saturated but contains a VM which is not saturated. Consequently, this VM can be migrated to another server that is not overload.

These three mechanisms of cloud architecture elasticity are formalized by the following rewrite rules.

3.2.1 Rewrite rules for Service/VM replication

Service replication is applied if the VM is not saturated and contains the requested service. However, the later has reached the authorized threshold of simultaneous requests.

```
crl [Replication-of-service] :
Replicate-service( S, SR)
< V : Vm | NbrService : SN , NbrReq : M ,
services : SS , Cout : C >
```

```
< S : Service | NbrClient : CN , Type : type>
=> < S : Service | >
< SR : Service | NbrClient : 0 , clients : empty , Type : type , Cout :Cout('Service ) >
< V : Vm | NbrService : (size(SS) + 1) , services : (add(SR, SS)) , Cout : C + Cout('Service ) >
if CN == MAXREQ /\ SN <= MaxService /\ S in SS = true /\ SR in SS = false .
```

The effect of the rewrite rule is to accumulate the cost of adding a service instance to the cost of the VM and to increment the number of services actually deployed in this VM. The newly created service is added to the list of services of the VM.

A VM replication is conditioned with the fact that the server is not saturated.

```
crl [Replcation-of-VM] :
Dup-VM( V, VV)
< V : Vm | state : loaded >
< E : Server | NbrVm : VN , vms : CL , state : non-loaded , Cout : C >
=> < V : Vm | >
< VV : Vm | NbrService : 0, services : empty , NbrReq : 80 , state : non-loaded , Cout : Cout('VM ) >
< E : Server | NbrVm : (size(CL) + 1) , vms : (add(VV, CL)) , state : non-loaded , Cout : C + Cout('VM ) >
if VN < MaxVM /\ V in CL = true /\ VV in CL = false .
```

The effect of the rewrite rule is to accumulate the cost of adding a new VM to the actual cost of the server. The number of VMs is also incremented and a new VM is created and added to the list of VMs of the server.

3.2.2 Rewrite rule for Service/VM consolidation

The Consolidation-of-service rewrite rule destroys useless (empty) copies of services/VMs when the workload scales down. The rule applicability conditions are the list of clients is empty and there exists another service of the same type that has not yet reached its maximal threshold of simultaneous requests.

```
crl [Consolidation-of-service] :
< V : Vm | NbrService : SN , services : SS , Cout : C >
< S : Service | Type : type , clients : CL >
< S' : Service | Type : type , NbrClient : 0 >
=>< S : Service | >
< V : Vm | NbrService : (size(SS) - 1) , services : (del(S' , SS)) , Cout : C >
if size(CL) > 0 /\ size(CL) <= MAXREQ /\ S in SS = true /\ S' in SS = true .
```

The effect of the rewrite rule is to decrement the number of services actually deployed on the VM and destroy the useless service.

The *Consolidation-of-VM* rewrite rule is similar to the Consolidation-of-service one expects that it operates on useless VMs instead of services.

3.2.3 Rewrite rule for VM migration

VM migration rule allows moving a VM from one server to another. The destination server might be not saturated. The rule execution effect consists of updating the destination server cost by accumulating the cost of a VM migration and incrementing the number of VMs. The user is added to the list of clients of both the service and the VM.

```
crl [Migrate-VM] :
allocate( U, S)
< U : User | connected : false>
< S : Service | NbrClient : CN , clients : LS >
< E : Server | vms : V VS , state : loaded >
< EE : Server | NbrVm : VN , vms : CL , Cout : C , NbrReq : Y , state : non-loaded >
< V : Vm | services : S SS , NbrReq : M , state : non-loaded >
=>< U : User | connected : true>
< S : Service | NbrClient : (size(LS) + 1) , clients : (add(U, LS)) >
< V : Vm | NbrReq : M + 1 , state : non-loaded >
```

```
< E : Server | NbrVm : (size(VS) - 1) , vms : (del(V, VS)) , state : non-loaded >
< EE : Server | NbrVm : (size(CL) + 1) , vms : (add(V, CL)) , NbrReq : Y + 1 , Cout : C + Cout('MigrateVm ) >
if VN <= MaxVM  /\ V in CL = false /\ CN <= MAXREQ /\ U in CL = false .
```

3.3 Elasticity Strategy Selector Formalization

In our probabilistic model, we associate to each rewrite rule; representing an elasticity strategy and already presented in the above sub-section, an applicability probability which is inversely proportional to the cost of the corresponding elasticity strategy. So, plus the cost is high, the rewrite rule is less probable to be applied. The probability of applying an elasticity strategy is also inversely proportional to the workload of the resources (the actual context), i.e., a resource is likely to be selected if it has less workload.

Before defining the strategies to control the various forms of elasticity, we start with quantifying the nondeterministic choice that exists in the cloud system environment. We initially define the strategy *RuleStrat* affecting a weight to each rewrite rule. For our Ticket Booking system, we have weighed the various elasticity rewrite rules with the corresponding elasticity strategy cost.

```
psdrule    RuleStrat    :=    given    state: CF:Configuration
is:     ( allocate-service ) -> 100 ;
        ( Consolidation-of-service ) -> 1 ;
        ( Replication-of-service ) -> 10 ;
        ( Replication-of-service2 ) -> 10 ;
        ( Replcation-of-VM ) -> 1 ;
        ( Consolidation-of-VM ) -> 1 ;
        ( Migrate-VM ) -> 1 ;
[none] .
```

Generally, the frequently solicited rewrite rule is the *allocate-service* one considering that the cloud system is dotted with a large capacity of resources and is able to treat a multitude of user requests simultaneously. So, we affect the highest weight, of value 100, to the allocate-service rewrite rule. A weight of 10 is affected to the Replication-of-service rewrite rule to treat requests for deploying new services or creating new instances of existing ones. The rest of the rewrite rules are rarely solicited and thus they are affected a weight of 1.

Given that CF is a variable defining the current configuration of the cloud system, the strategy of weighting the various elasticity rewrite rules is applied at each possible state. Thus, the *RuleStrat* strategy quantifies the nondeterministic choice of the rule to be applied at each stage of the execution.

In what follows, we define the strategies of context and substitution that are common to all rewrites rules. We only present elasticity rules that are subject to non-determinism as the replication and migration rules while selecting the resource to be used. We assign a weight to the context strategy *CtxStrat* which represents the probability of selecting and applying Replication-of-service rule. The weight is inversely proportional to the cost of creating a new instance of a service.

The substitution strategy *SubStrat* is uniform since there is a unique correspondence substitution.

```
------Ctx et Sub : Replication of Service ----------
 psdcontext    CtxStrat    :=    given    state: CF:Configuration < V : Vm | Cout : C >
rule: Replication ice -of-serv
is: ([] < V : Vm | Cout : C > Cout('Service ) >) -> ( 1 / ( C + Cout('Service )))
[none] .
psdsubst    SubstStrat    :=    given    state: CF:Configuration
                                    rule: Replication-of-service

context: CTX:Configuration
is: uniform
[none] .
```

The context and substitution strategies for replicating a VM are defined in a similar way.

The context in which the *Migrate-VM* rewrite rule can be applied depends on the probability associated to the cost of migrating a VM. The selection of the destination server is non-deterministic. Thus, the

substitution strategy intervenes and selects the server actually having the less workload.

The less loaded server is identified by affecting a weight of *(1/(XY + size (XCL)))* to the substitution strategy. Such weight is inversely proportional to the sum of VMs actually deployed on the server and the number of requests.

```
---------------Ctx et Sub : Migrate VM ----------
psdcontext CtxStrat := given state: CF:Configuration
< XEE : Server | NbrVm : XVN , vms : XCL , Cout : XC >
rule: Migrate-VM
is: ( [ ] < XEE : Server | >)
->(1 / XC + Cout('MigrateVm )))
[none] .
psdsubst SubstStrat := given state: CF:Configuration
< XEE : Server | NbrVm : XVN , vms : XCL , Cout : XC , NbrReq : XY >
rule: Migrate-VM
context:
CTX:Configuration
is: { EE <- XEE , VN <- XVN ,CL <- XCL ,C <- XC } -> ( 1 / (XY + size(XCL)))
[none] .
```

For the rest of the rules, strategies for context and substitution are uniform and are randomly selected (random) at any state.

Back to our Ticket Booking system, we present two scenarios of dynamic resizing of the system.

The first scenario consists of creating a new *VM* whenever the *VM1* becomes saturated. The result is shown in Figure 1.

The second scenario concerns the virtual machine migration strategy. We consider that client *A2* is actually soliciting the Air Ticket Booking service *S1* which is deployed on *VM1*; not yet saturated. But *Server1*, containing *VM1*, is no more able to treat new requests since the *NBReq* has reached is maximal threshold of 100 simultaneous requests.

```
unbounded probabilistic rewrite of term: CloudConf
in Cloud-Specification using probabilistic strategy CloudStrat

Rules applied: saturate-vm Replcation-of-VM

Error: Reached an error (kinded) state!

Result Configuration: < 'A1 : User | connected : true > < 'A2 : User |
    connected : true > < 'DataCenter1 : DataCenter | loadbalancer : 'LoadB1,
    servers :('Server1 'Server2 'Server3)> < 'LoadB1 : LoadBalancer | connected
    : true > < 'S1 : Service | Cout : 10,NbrClient : 3,Type :
    'AirTicketBoooking,clients :('A1 'A2 'B)> < 'Server1 : Server | Cout : 20,
    NbrReq : 1,NbrVm : 3,state : non-loaded,vms :('VM1 'VM2 'VV)> < 'VM1 : Vm |
    Cout : 10,NbrReq : 1,NbrService : 3,services :('S1 'S2 'S3),state : loaded
    > < 'VV : Vm | Cout : 10,NbrReq : 0,NbrService : 0,services : empty,state :
    non-loaded >
```

Figure 1: VM Replication

In this situation, the VM migration strategy is selected by the Elasticity Strategy Selector and applied to move *VM1* to a less loaded server. A set of steps is applied to allocate service S1 to user *A2*. As a first step, the state of *Server1* is updated and becomes loaded. Then, the destination server is selected via the substitution strategy of the Migrate-VM rewrite rule and *VM1* is moved. Finally, service S1 is allocated to user *A2*. The destination server selection mechanism is non-deterministic since two servers (*Server2* and *Server3*) are actually potential candidates to receive *VM1*. The non-determinism is resolved by the substitution strategy *SubstStrat* (see Figure 2) which selects the less loaded server since server selection weight is inversely proportional to the sum of its VMs and requests. The substitution strategy selects Server3. The result of migrating *VM1* from *Server1* to *Server3* is shown in Figure 2.

```
one-step probabilistic rewrite of term: CloudConf
in Cloud-Specification using probabilistic strategy CloudStrat

Rule applied: Migrate-VM

Result Configuration: < 'A2 : User | connected : true > < 'DataCenter1 :
    DataCenter | loadbalancer : 'LoadB1,servers :('Server1 'Server2 'Server3
    < 'LoadB1 : LoadBalancer | connected : true > < 'S1 : Service | Cout : 1
    NbrClient : 3,Type : 'AirTicketBoooking,clients :('A1 'A2 'B)> < 'Server
    Server | Cout : 10,NbrReq : 100,NbrVm : 1,state : loaded,vms : 'VM2 > <
    'Server2 : Server | Cout : 10,NbrReq : 60,NbrVm : 1,state : non-loaded,v
    : 'VV > < 'Server3 : Server | Cout : 40,NbrReq : 26,NbrVm : 2,state :
    non-loaded,vms :('V 'VM1)> < 'VM1 : Vm | Cout : 10,NbrReq : 2,NbrService
    2,services :('S1 'S2),state : non-loaded >
```

Figure 2: Migrating VM1 from Server1 to server2

4 Elastic Cloud Analysis

To ensure the required quality of service of the proposed model for elastic cloud systems, we are interested with checking certain non-functional properties i.e. quantitative. We have opted for calculating the rate of resources implied in the treatment of the customer's requests or calculating the effort supplied by the cloud system to allocate a service to a given user. Such effort may require the deployment of new services, the creation of new machines, and so on. The effort is calculated on the basis of the workload of the virtual machine VM; actually containing the required service and running on a physical server S according to the following formula:

```
Effort(S) = (Initial_Effort(S) + SUM_VMs(S) + SUM_Service(S)) /
(SUM_Requests(VM)+SUM_Request(S))
```

For efficiency reasons, rather than calculating the effort function for each intermediate state to be reached by the rewriting engine, as defined above, we calculate a cumulated effort and include the effort function *eff(e)* as a term in system state. Back to our Ticket Booking system to estimate the effort concerted to allocate the Air Ticket Booking service *S1* to user *A2*. Initially, the effort is zero. It is then augmented according to the Effort function defined above. We define a parameterized predicate *EffectiveCloud*, in a predicate module CLOUD-PRED. The semantics associated to the *EffectiveCloud* predicate is the following: the predicate is true in a given state *S* if the cumulative effort is strictly lower than a threshold *K*. The CLOUD-PRED module bellow defines the desired predicate:

```
(spmod CLOUD-PRED is
protecting Cloud-Specification .
--- sort for system states
smcstate Configuration .
var Effort : Rat .
var CF : Configuration .
var K : Rat .
--- declaring the parametric state predicate
psp EffectiveCloud : Rat .
--- defining its semantics
csat (CF eff(Effort)) |= EffectiveCloud(K)
if Effort < K .
endspm)
```

We can now check that the system verifies the security property which is ensured if the cumulated effort of the cloud system while allocating a service never exceeds a threshold K with a probability of at least 0.9. This property ensures the desired QoS of cloud services, i.e., minimizing the effort (*SUM_resources/SUM_requests*), the cost is also reduced and even the workload. Indeed, while carrying the following PTCL property, the simulation result is illustrated in Figure 3. The TicketBooking system is able to allocate services with an effort of K = 5 in 94% of the cases.

```
(smc CloudConf |= P >= 0.9 [G EffectiveCloud(5)] using CloudStrat .)
```

5 CONCLUSION

The goal of managing cloud resources capacity is to ensure service availability in spite of the abundance of user requests but still with the desired quality of service. Guarantying service availability and at the same time optimizing resource utilization has inspired a multitude of research work on the cloud elasticity property using probabilistic models. However, a significant effort is necessary to manage, plan elastic cloud systems and automatically identify the number of resources to be added or removed. Thus adjusting system capacity to a model of workload variations requires a great expert testimony.

Our contribution consists of proposing a formal approach for specifying cloud architecture, its dynamics and elasticity policies in terms of resource provision and release. Indeed, we treat and resolve the problem of non-determinism caused by the applicability of various elasticity actions and the availability of several resources of the same nature. We have quantified the non-determinism to obtain a fully probabilistic model by associating a cumulative cost attribute to each cloud resource. Then, a dynamic weighting of the elasticity strategies is performed to select the more adequate one. Interesting results have been obtained while performing the probabilistic analysis of the elastic cloud system to evaluate the effort furnished by the cloud system to ensure service availability.

As future work, we intend to ensure an automatic code generation to directly obtain executable PSMaude

specifications from models instantiating the defined meta-model. We also envisage developing an integrated framework for specifying self-adaptive systems in general and elastic cloud systems in particular.

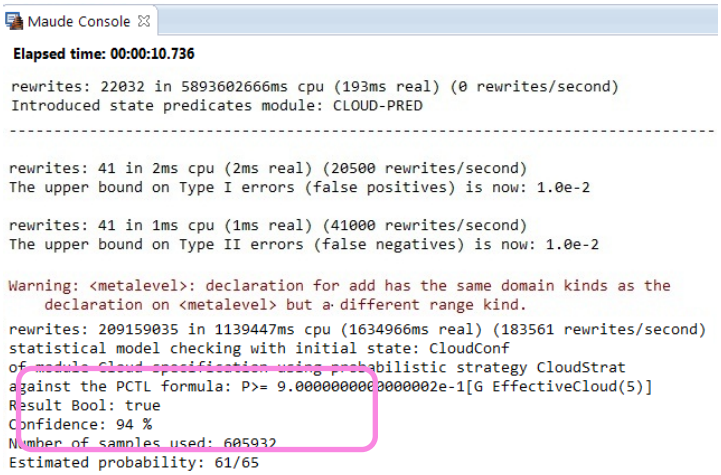

Figure 3 : Simulation result of the effort function simulation

References

[Ben 13] L. Bentea, P.C., Olveczky. A probabilistic strategy language for probabilistic rewrite theories and its application to cloud computing. In WADT'12, LNCS, vol. 7841, pp. 77-94. Springer, 2013.

[Cla 08] M. Clavel, F. Duran, S., Eker, P., Lincoln, N., MartiOliet, J., Meseguer, and C., Talcott. Maude Manual (version 2.4)', SRI International, 2008.

[Gee 09] J., Geelan, M., Klems, R., Cohen, J., Kaplan, D., Gourlay, P., Gaw, D., Edwards, B. de Haaff, B., Kepes, K., Sheynkman, O., Sultan, K., Hartig, J., Pritzker, T., Doerksen, T. von Eicken, P. Wallis, M., Sheehan, D., Dodge, A. Ricadela, B., Martin, B. Kepes, B., and I. W., Berger. Twenty-One Experts Define Cloud Computing. ICSE Workshop on Software Engineering Challenges of Cloud, 2009.

[Joh 11] K., Johnson, S., Reed, and R., Calinescu, Specification and Quantitative Analysis of Probabilistic Cloud Deployment Patterns. Hardware and Software: Verification and Testing - 7th International Haifa Verification Conference, {HVC} 2011, Haifa, Israel, Revised Selected Papers, pp. 145-159, 2011.

[Li 16] Ai, W., Li, K., Lan, S., Zhang, F., Mei, J., Li, K., and R. Buyya. On Elasticity Measurement in Cloud Computing', Scientific Programming, Vol. 2016, Article ID 7519507, 13 pages, 2016.

[Mes 93] J., Meseguer. A Logical Theory of Concurrent Objects and its Realization in the Maude Language. Research Directions in Object-Based Concurrency, MIT Press, pp. 314-390, 1993.

[Ron 14] M. Rong. Modeling and analysis BPEL-based web services composition using XYZ. The 9th International Conference on Computer Science & Education (ICCSE 2014). Vancouver Canada, pp.1083–1088, 2014.