Model Checking of Cost-Effective Elasticity Strategies in Cloud Computing

Rawand Guerfel[1](✉), Zohra Sbaï[1,2], and Rahma Ben Ayed[1]

[1] Université de Tunis El Manar, École Nationale d'Ingénieurs de Tunis, BP. 37 Le Belvédère, 1002 Tunis, Tunisia
{Rawand.Guerfel,zohra.sbai,rahma.benayed}@enit.rnu.tn

[2] College of Computer Engineering and Science, Prince Sattam Bin Abdulaziz University, PO Box 151, Al-Kharj 11942, Kingdom of Saudi Arabia

Abstract. Cloud computing is a revolution in how computing power is delivered to business. It offers different measured services to clients who require them by writing a simple request. These requests are becoming more and more complex so that services need to be composed to meet them. Additionally, these Cloud composite business services (CCBSs) need to be elastic, i.e. their number should be replicated or reduced according to the number of their user demands. Ensuring these two operations is done according to a well-defined strategy. We are interested in this paper in cost-effective elasticity one. Applying this strategy on CCBSs gives birth to a system that needs to be checked to insure that SLA constraints, such as deadline specified by the user, are not violated. In this paper, we present a formal model using Timed Coloured Petri nets to model, check and compare between these strategies before implementing them in real Cloud.

Keywords: Cloud computing composition · Cloud elasticity
Cost-effective strategy · SLA · Formal model
Timed Coloured Petri Net

1 Introduction

Cloud computing is a as a service model where different resources and data such as servers, switches, storage, applications and services are accessed over the internet. It is a model that enables ubiquitous on demand access to a shared pool of configurable computing resources which can be rapidly provisioned and released with minimal management effort.

There are basically three layers to the Cloud that are used differently based on what they offer. The first layer is Infrastructure as a Service (IaaS) which offers virtual systems that can be connected using internet. The second layer is Platform as a Service (PaaS) which is a proof model for running applications without the hassle of maintaining the hardware and software infrastructure at

L. Braubach et al. (Eds.): ICSOC Workshops 2017, LNCS 10797, pp. 80–92, 2018.
https://doi.org/10.1007/978-3-319-91764-1_7

the company. The last layer is Software as a Service(SaaS) which is a delivering way of application as a service. Using this layer, one is not obliged to install and maintain software [13].
Nowadays, user requirements are becoming more and more complex that sometimes, a single Cloud business service cannot meet user requests. It needs to communicate and to be combined with other business services to respond to user demands. In this case, we are treating the composition mechanism in Cloud computing, an already discussed issue in our previous works [3,4].

The challenge with these composite Cloud services is that they should be elastic [6]. Indeed, elasticity is one of the most important characteristic that Cloud computing offers to its users. More precisely, it allows the providers to adapt in term of numbers to the user demands in a transparent way. We distinguish two types of elasticity. The first type is vertical elasticity which is related to the scaling up or down of resources of a specific Cloud service without modifying its number (e.g.: the power or the capacity of servers). The second type is horizontal elasticity which is related to the removing or the adding of Cloud service instances. In our work, we are interested in horizontal elasticity. More precisely, in horizontal elasticity type, when the number of user demands for the CCBS increases, the provider has to replicate as many copies of service instances as this number. Similarly, when the number of user demands decreases, the provider has to delete the unused copies of service instances. Replicating and deleting actions are done according to an elasticity-strategy.

Executing elasticity strategies on composite Cloud business service does not necessarily imply the replication or the deleting of the whole composite service. Indeed, according to some indicators, we can apply these operations on only some of the services involved in the composition. This gives birth to a system composed of different services that interconnect and communicate between each other. Users access should be well organised so that the system will not suffer from some problems such as deadlock, conflit, etc.

Let's also note that when applying and choosing an elasticity strategy, one has to take into account two major factors which are: the deadline parameter specified by the user and the gained cost for the Cloud provider. Indeed, on the one hand, SLA has to be ensured. More particularly, when applying an elasticity strategy, we have to check that the deadline is not violated so the provider does not pay a penalty. On the other hand, we should make sure that the elasticity maximizes the cost gained by the provider. This leads to the use of what is known by "cost-effective elasticity strategy".

It is in this context that our work is oriented. Indeed, we propose to check the validity of the model obtained when executing cost-effective elasticity strategies on composite Cloud business services. This model should ensure the non-violence of the user deadline constraint. To do so, we use formal modelling. More precisely, we use Coloured Petri nets (CPN) [8]. CPN allow us to assign time and data information to each service, so that we can assign cost to each service. Besides, CPN offers us the possibility to differentiate between multiple users. Indeed, coloured tokens are associated with data that could be used as specific ID for each user.

The reminder of this paper is structured as follows. We start the Sect. 2 with presenting the system model by giving some important definitions. Then, we give a motivation example based on two existing strategies. Then, we move to the Sect. 3 to model the elasticity strategy using CPN tool [14]. Also, in this section, an algorithm of generation of CCBS model in CPN is detailed. To valid this model, we check some properties detailed in Sect. 4. Afterwards, we refer, in Sect. 5, to related works. In Sect. 6, we present conclusions and expose some future works.

2 Cost-Effective Elasticity Strategy of Composite Cloud Business Services

Cloud services should be characterized by one of the most important benefic offered by the Cloud computing which is: elasticity. In fact, by rapid elasticity, the Cloud can dynamically allocate or deallocate resources based on the customer configurations [6]. As we already mentioned, this work focuses on horizontal elasticity.

Horizontal elasticity of composite Cloud business services means adapting the number of this composite service to the number of user demands. Many indicators exist to help the provider know when to apply the elasticity strategy namely: the number of user demands, the maximum/minimum number of active sessions, number of user request per unit of time, etc. In our work, we are interested in elasticity strategies that are based on these two indicators:

- the maximum and minimum number of user demands that each service can hold.
- the cost gained when applying the elasticity strategy.

2.1 System Model

Ensuring the elasticity of composite Cloud business services is an important and necessary step. To do so, many strategies have been proposed. To execute the necessary actions, most of them are based on the number of users accessing the CCBS as an elasticity indicator. However, we are interested in ones that add the cost factor when deciding to process these actions.

In this section, we give some important definitions.

Definition 1 (CBS definition): *A CBS_i is defined with the tuple* $(name_i, max\text{-}thres_i, min\text{-}thres_i, Resp\text{-}T_i, cost_i, Rep\text{-}Cost_i)$ *where:*

- $name_i$*: is the name of* CBS_i
- $max\text{-}thres_i$*: is the maximum threshold of* CBS_i
- $min\text{-}thres_i$*: is the minimum threshold of* CBS_i
- $Resp\text{-}T_i$*: is the response time of* CBS_i

- *$cost_i$: is the cost of the CBS_i*
- *Rep-$Cost_i$: is the cost of replication action of the CBS_i*

Definition 2 (CCBS definition): *A $CCBS_0$ is a combination of n CBS_{oi} ; where $i \in 1..n$; and has the following structure:*

$$\begin{aligned} CCBS_0 &= (CBS_{01}, CBS_{02}, ..., CBS_{0n}) \\ &= (< name_{01}, max - thres_{01}, min - thres_{01}, Resp - T_{01}, cost_{01}, \\ &\quad Rep - Cost_{01} >, ..., < name_{0n}, max - thres_{0n}, min - thres_{0n}, \\ &\quad Resp - T_{0n}, cost_{0n}, Rep - Cost_{0n} >) \end{aligned}$$

Definition 3 (CCBS's response time): *The response time of one $CCBS_i$, noted by Resp-T_i, is the sum of the response time of all its CBSs. It is defined as follows:*

$$Resp{-}T_i = \sum_{j=1}^{n} Resp{-}T_{ij} \ ;$$

Where n is the number of CBSs that compose the CCBS.

Before accessing these services, a SLA is defined between the user and the provider. In this SLA, users specify some constraints that should be valid such as: the availability, the deadline, the budget, the penalty, etc. Some of these parameters are treated in this paper and detailed in the following definition.

Definition 4 (User requirement): *A user requirement is given as follows: UR=(name, dead, budget, penalty); knowing that:*

- *Name: is the name of either the CCBS or the CBS. In fact, the user can access to just one Cloud business service.*
- *Dead: is the maximum time given to the provider to respond to the user requirement.*
- *Budget: is the price offered to the provider if the service is given before Dead.*
- *Penalty: is the penalty to be paid by the provider if the request is given after Dead, the parameter specified by the user.*

2.2 Motivation

Let's suppose that we have four Cloud business services CBS_{11}, CBS_{12}, CBS_{13} and CBS_{14} composing the $CCBS_1$.

Each CBS is defined as follows:

$CBS_{11} = (CBS_{11},\ 30,\ 5,\ 0.3,\ 0.2,\ 0.25)$
$CBS_{12} = (CBS_{12},\ 35,\ 5,\ 0.4,\ 0.35,\ 0.45)$
$CBS_{13} = (CBS_{13},\ 40,\ 3,\ 0.5,\ 0.55,\ 0.7)$
$CBS_{14} = (CBS_{14},\ 21,\ 6,\ 0.2,\ 0.15,\ 0.2)$

Let's suppose that this CCBS is required by multiple users in the same time. So, an elasticity strategy must be applied.

Let's suppose that at time t1, 20 users want to access to CCBS. Note that the deadline of 2 users are not respected.

Then, at time t2, 20 other users want to access to CCBS. All deadlines are respected but only 15 users are a cost gain for the provider perspective.

Finally, at time t3, 25 users leave.

To make the necessary decision, an elasticity strategy has to be applied. In fact, our work is based on two essential elasticity strategies detailed in the following.

Elasticity strategy 1 [5]**:**
First of all, the provider checks in every unit of time (e.g. second) the maximum and the minimum threshold of each CBS. If the maximum one is reached, then, he calculates the cost benefit when applying a replication action. If the cost is a gain for the provider perspective, then, a replication action is processed. Else, he waits for more users having a higher budget. Let's note that if the minimum threshold of one CBS is reached, then, a deletion action of this service is automatically done. Indeed, the provider has nothing to lose when executing this action. In this strategy, they supposed that all deadlines specified by users are already checked.
Applying this strategy to the previous scenario gives the following result :
At time t1,accept only 18 users. Here, no service will be replicated.
At time t2, accept only 15 users. In this case, CBS_{11} and CBS_{14} reach their maximum thresholds and have to be replicated to CBS_{21} and CBS_{24}.
At time t3, eliminate 25 users. CBS_{21} and CBS_{24} reach their minimum thresholds and have to be removed and replaced by CBS_{11} and CBS_{14}.

Elasticity strategy 2 [7]**:**
In this strategy, the provider checks in every unit of time if the deadlines specified by users are respected. If 90% of users have a response time lower than their deadlines, so, accept the other 10% of users whatever the penalty to be paid and do the replication action. Else, if less than 90% have a response time lower than their deadlines, the provider does not accept them and waits for other users that satisfy this condition.
Applying this strategy to the previous scenario gives the following result:
At time t1,accept 20 users. Here, no service will be replicated.
At time t2, accept all the 20 users. In this case, CBS_{11}, CBS_{12} and CBS_{14} reach their maximum thresholds and have to be replicated to CBS_{21}, CBS_{22} and CBS_{24}.
At time t3, eliminate 25 users. CBS_{21}, CBS_{22} and CBS_{24} reach their minimum thresholds and have to be removed and replaced by CBS_{11}, CBS_{12} and CBS_{14}.

3 Formal Modeling of Elasticity of CCBS

The specification of complex systems play an important role in their reliability control. Indeed, they serve as reference for system implementation. The use of formal methods [2] is then the best way to assist the design and validation of these specifications.

More specifically, we use CPN which are an extension of Petri nets(PN). Indeed, PN are used to model the dynamic behavior of discret systems. They are composed of two types of objects which are: places, that represent the states of the system and contain information represented by tokens, and transitions which represent the events of the system. Places and transitions are related by arcs. However, with PN, it is impossible to model similar behaviors using a single condensed representation. This limitation of PN does not allow us to differentiate between users. That's why we use CPN.

Actually, CPN offer three types of extension which are: extension with time, extension with data and extension with hierarchy. Since we can represent users with tokens, the extension with data allows us to assign information specified by users in their requirements. The extension with time allows to assign a commun time to the group of users demanding the CCBS at the same moment. This helps us organize and differentiate between different users.

Below is the formal definition of CCBS using CPN.

Definition 3: *$CCBS_i$ is a CPN (P, T, C, E, M_0) where:*

- *P is the set of places. They represent the states of the CBSs composition.*
- *T is the set of transitions. Each transition represents a CBS.*
  Note that: $P \cup T = \phi$ and $P \cap T = \phi$
  Each transition has a specific price (the cost of the service). So, we assign to each transition a variable pr_i indicating its price if it is not replicated, else, the price of its replication.
  For example, the price of the transition T_{i1} is defined as follows: $val\, pr_{i1} = 0.4$;
- *C is the set of colours. It defines for every place its colour domain. In our proposal, every place has as type "Info" defined as follows:*
  - *colset Info = product U*RE*RE timed; knowing that:*
  - *colset U = index us with 0..n; : every token belonging to $CCBS_i$ has the value us(i). us(i) represents the set of users that can be handled by one $CCBS_i$.*
  - *colset RE = REAL; : an integer type. The first RE represents the number of users that can be handled by one CCBS, and the second one represents the sum of their budgets.*

  Let's note that Info is timed to indicate the evolution of the process through time. For example, the token (u(1), 25, 26)@1 indicates that at time 1, we have 25 users whose budget is 26.
- *E is an arc expression function. The colour of each arc must be the same colour of the place to which the arc is entering or outgoing.*

It is defined as follows: (us,nb,bd) knowing that these variables are declared as follows:
var us: U; var nb,bd: RE; us, nb and bd represent respectively users ID, their number and their budgets.

- M_0 *is the initial marking of the net. It describes, in a net, how coloured tokens are situated in different places at a specific time of the execution.*

Notation: *Let N be a CPN representing* $CCBS_i$, $p \in P, t \in T$ *and k is the defined arc colour. We note:*

- $^{\bullet}t = \{p \in P \mid W^{-} > 0\}$*: Input places of t.*
- $t^{\bullet} = \{p \in P \mid W^{+} > 0\}$*: Output places of t.*
- $Pre : P \times T \rightarrow \{k, 0\}$. *If an arc links* P_m to T_n*, then,* $Pre(P_m, T_n) = (us, nb, bd)$*, else,* $Pre(P_m, T_n) = 0$.
- $Post : P \times T \rightarrow \{k, 0\}$. *If an arc links* T_n to P_m*, then,* $Post(P_m, T_n) = (us, nb, bd\text{-}pr_{in} * nb)$*, else,* $Post(P_m, T_n) = 0$.

After having defined the modeling of each CCBS, we move now to explain the modeling of elasticity strategies, using the Algorithm 1.

In fact, when a transition T_{ik} of one $CCBS_i$ is not replicated and used from another $CCBS_j$, i.e. $T_{jk} \leftarrow T_{ik}$, then, two cases exist:

- T_{ik} is the first transition of the $CCBS_i$, i.e., $T_{ik} \leftarrow T_{i1}$. In this case, we create a transition named $Tlink_j$ having as input place P_{i0} of $CCBS_i$ and output place P_{j0} of $CCBS_j$. Then, use the non-replicated transition T_{j1} from $CCBS_j$ and link it to the input places of the next transition of $CCBS_i$. This is given by the steps 6–17 in the Algorithm 1.
- T_{ik} is any transition of the $CCBS_i$, except the first one. In this case, we link the previous transition of T_{ik} to the input places of T_{jk}. Then, T_{ik} is linked to the input places of the next transition of T_{ik}, which is $T_{i(k+1)}$. Steps 24 to 36 of the Algorithm 1 model this replication.

Let's note that the replication of only one transition requires the creation of a new CCBS.

The deletion action of just one transition implies the deletion of its input places. Its output places will be linked to the same non-replicated transition. However, if all the CCBS will be deleted because of the decrease in number of user demands, then, all its transitions and places will be deleted and the rest of users will be assigned to the previous existing CCBSs. By applying this algorithm, the modeling of the strategies 1 and 2 using CPN Tool is represented by the Fig. 1.

4 Verification of Elasticity Strategies

4.1 Formal Analysis of the Proposed System Modeling

The use of formal methods for software and hardware design is motivated by the welling to achieve the appropriate mathematical analysis, which can contribute to the reliability and robustness of a design. This guarantees safe operation of these critical designs. Indeed, the first question that may arise after designing our model is if this model checks the specification and if it is correct and coherent. It therefore seems essential to check our graph. To do so, we use formal verification.

Indeed, our composition model is a combination of many CCBSs. Each one of them is a CPN having a specific initial marking. A case in one CCBS starts with a token located in the initial place. After a series of steps, this token evolves towards a final marking that should be located in the final place. So, an important property to be checked in this model is the reachability property. In fact, we must ensure that from such an initial marking, it must be possible to reach the final place. This is what we call the **reachability** property of the marking of output places from the marking of input places [11].

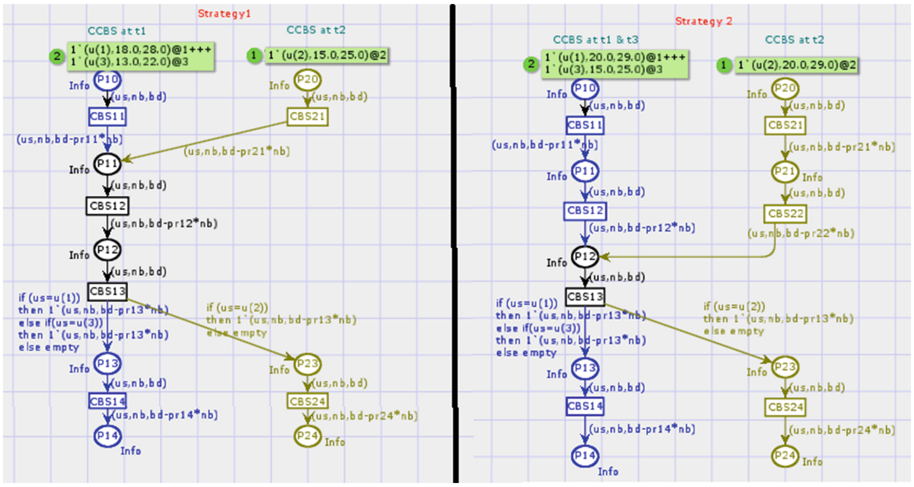


Fig. 1. Modeling of elasticity strategy using CPN tool

Moreover, we should check the absence of dead transitions in every CCBS. That is to say, all transitions can be enabled. This is called the **absence of deadlock** property [12].

In order to express specific properties and verify them, first of all, a generation of state graph must be processed. CPN tool automatically generates and calculates this state graph using a strongly connected components (SCC) graph. Then, we can express properties and query them so that CPN tool checks them. In our case, two properties are checked which are:

Algorithm 1. Modelling of CCBSs replication using CPN

```
 1: for $CCBS_i$ in CCBSLi do
 2:     $P_{i0} \leftarrow P_{10}$;
 3:     $P_{im} \leftarrow P_{1m}$;
 4:     $\sum_{k=1}^{p} T_{ik} =^{\bullet} P_{i0}$
 5:     for j=1 to p do
 6:         if $(T_{ik} \in S2)$ then
 7:             $Pre(P_{i0}, T_{ik}) = (us, nb, bd)$;
 8:         else if ($T_{ik}$ is used from $CCBS_l$) then
 9:             $Pre(P_{i0}, Tlink_l) = (us, nb, bd)$;
10:             $Post(P_{k0}, Tlink_l) = (us, nb, bd)$;
11:             $T_{ik} \leftarrow T_{lk}$;
12:             $T_{lk}^{\bullet} \leftarrow (T_{lk}^{\bullet} +^{\bullet} T_{i(k+1)})$;
13:             $Post(^{\bullet}T_{i(k+1)}, T_{lk}) = (if\ \ us = u(i)\ then\ 1`(us, nb, bd\text{-}pr_{lk} * nb)$
14:             $else\quad empty)$;
15:             $Post(^{\bullet}T_{l(k+1)}, T_{lk}) = (if\ \ us = u(l)\ then\ 1`(us, nb, bd\text{-}pr_{lk} * nb)$
16:             $else\quad empty)$;
17:         end if
18:     end for
19:     for k=(p+1) to n do
20:         if $(T_{ik} \in S2)$ then
21:             $T_{ik} \leftarrow$ Replication of $(T_{lk})$;
22:             $^{\bullet}T_{ik} \leftarrow$ Replication of $(^{\bullet}T_{lk})$;
23:             $Pre(^{\bullet}T_{ik}, T_{ik}) = (us, nb, bd)$;
24:         else if ($T_{ik}$ is used from $CCBS_l$) then
25:             $T_{i(k-1)}^{\bullet} \leftarrow^{\bullet} T_{lk}$;
26:             $Post(^{\bullet}T_{lk}, T_{i(k-1)}) = (us, nb, bd\text{-}pr_{i(k-1)} * nb)$;
27:             $T_{ik} \leftarrow T_{lk}$;
28:             if $(T_{lk}^{\bullet} != P_{lm})$ then
29:                 repeat step 12-16;
30:             else
31:                 $T_{lk}^{\bullet} \leftarrow (T_{lk}^{\bullet} + P_{im})$;
32:                 $Post(P_{im}, T_{lk}) = (if\ \ us = u(i)\ then\ 1`(us, nb, bd\text{-}pr_{lk} * nb)$
33:                 $else\quad empty)$;
34:                 $Post(P_{lm}, T_{lk}) = (if\quad us = u(l)\ then\ 1`(us, nb, bd\text{-}pr_{lk} * nb)$
35:                 $else\quad empty)$;
36:             end if
37:         end if
38:     end for
39: end for
```

- Reachability: This property consists in checking if all output places are reachable. Indeed, the reachability of output places confirms that the process of each CCHS is successfully done. CPN tool offers us a simulation palette to check the execution of different tokens situated in the initial places. Our model was simulated more than 100 times to check if output places are reached. All of these simulations have showed the success reachability of the four output places.

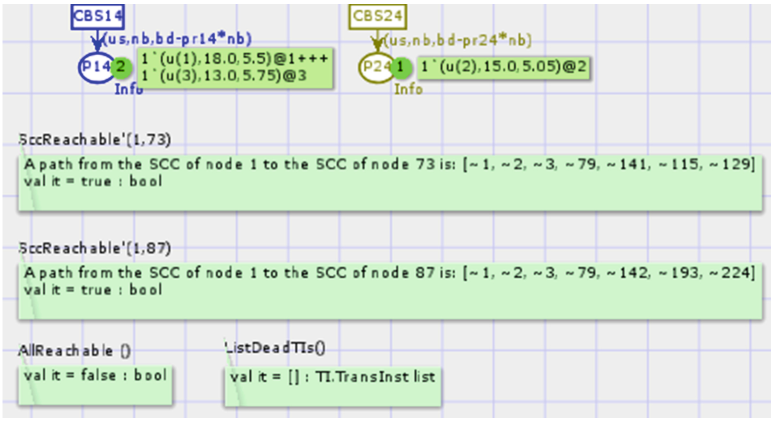
However, this not enough to confirm that output places are always reachable from the initial marking. So, a query for each output place was executed to confirm this property. The expression of this query is as follows: **SccReachable'(p1,p2)** ; knowing that:

p1: is the initial state of the model, i.e., when input places are marked.

p2: is the final state of a specific CCBS, where the output place of CCBS is marked and all other places do not contain the token of the current CCBS. This query returns either false is the final node is not reachable or true with the specific path if the final node is reachable.

The state of different places is detected from the SCC graph. Let's note that the initial state is 1 and the final states of CCBS1, CCBS2 are respectively: 73 and 87 So, two different queries must be checked which are: SccReachable'(1,73) and SccReachable'(1,87). Both of these two queries were successfully checked and the result is shown in Fig. 2.

- Absence of deadlock: A deadlock corresponds to a CPN marking in which no more transition is allowed. So, there must be no dead transitions. This can be checked in CPN tool using the following query: **ListDeadTIs()**. When executing this query, we can have two possible results: even a list of dead transitions or an empty list. In our model, this query returned an empty list, as shown in Fig. 2, a result that confirms the non-existence of deadlock.



```
1`(u(1),18.0,5.5)@1+++
1`(u(3),13.0,5.75)@3

1`(u(2),15.0,5.05)@2

SccReachable'(1,73)
A path from the SCC of node 1 to the SCC of node 73 is: [~1, ~2, ~3, ~79, ~141, ~115, ~129]
val it = true : bool

SccReachable'(1,87)
A path from the SCC of node 1 to the SCC of node 87 is: [~1, ~2, ~3, ~79, ~142, ~193, ~224]
val it = true : bool

AllReachable ()
val it = false : bool

ListDeadTIs()
val it = [] : TI.TransInst list
```

Fig. 2. Formal verification of CCBSs using CPN tool

Thus, we can confirm that the proposed modelling is valid. It does not contain any deadlock and reaches always the final states.

5 Related Work

Cloud Elasticity is a highly studied topic. Several mechanisms have been proposed to ensure it. However, we focus on works which were proposed to minimize the cost when applying elasticity.

Liu et al. [9] proposed an algorithm allowing them to minimize the cost of used SaaS by decreasing the unused Virtual machines from the IaaS. This algorithm aims at not violating the performance provided to the final user.

Wu et al. [15] proposed a system that maximizes the accepted number of users requesting a certain SaaS. This is done by an efficient placement of requests on Virtual machines offered by different IaaS providers. In fact, they proposed an algorithm that maximises the use of already initiated VMs so that many users can access them after being classified according to the waiting time. This solution is a cost benefic for the provider perspective.

Han et al. [5] have also focused on cost reduction when applying an elastic scaling approach of multi-tier applications in Cloud computing. Indeed, they proposed an approach that detects the bottlenecks in a class of these applications so that they can accordingly scale up and down resources at these points.

The above cited works are dealing with the vertical elasticity whereas we are handling horizontal one. Besides, the proposed models were not checked, which is a very important task to ensure their validity.

However, Narkos et al. [10] proposed a model, based on Markov Decision Chain, allowing the automatic elasticity by increasing and decreasing the number of Virtual machines. Indeed, the action that should be processed to ensure the elasticity operation is checked and expressed using PCTL. Besides, the reachability property is checked in this model using PRISM tool. This work treats also the vertical elasticity, which is not the case for our work. Moreover, we are handling the composite Cloud services and not the atomic Cloud services.

It is in this sens that the work of Amziani et al. [1] is oriented. Indeed, they proposed a controller to check the behavior of service-based business process in the Cloud when applying elasticity operations. The controller is modeled by high level Petri nets. Time and maximum and minimum thresholds of one service are the main indicators of elasticity actions. Their work was a start point for us but our work differs from them in three main points which are:

- Additionally to the threshold, our approach focuses on cost when applying elasticity actions.
- Our approach checks the validity of the obtained model before comparing between strategies.
- The modeling that we propose allows even the modeling of just one Cloud service in case it is required atomically.

6 Conclusion

Cloud computing has proven to be more secure, more reliable, more scalable and more affordable than traditional IT. These are some of the Cloud characteristics that made it widely used in many fields. As a result, many sectors of this field used the Cloud architecture to offer their services that must be composed to perfectly meet the demand of their users, whose number is more and more increasing. So, a replication of the composite service, called CCBS, must be processed to answer to user queries at the same time. Some strategies were proposed in this context. Our contribution in this paper was to check the validity of these strategies using formal models and to compare between them. To do so, our composition was modeled using CPN and was validated by checking the reachability and the absence of deadlock properties.

However, CPN tool does not allow us to check specific properties. So, as a future work, we propose to check soundness and temporel properties using a suitable tool. Besides, we intend to test this mechanism with real CBSs and to implement a tool allowing the automatic execution of elasticity actions.

References

1. Amziani, M., Melliti, T., Tata, S.: Formal modeling and evaluation of service-based business process elasticity in the cloud. In: 22nd IEEE International Conference on Collaboration Technologies and Infrastructure (WETICE 2013), pp. 284–291. Hammamet, Tunisia, June 2013
2. André, P.: Methodes formelles et a objets pour le developpement du logiciel: etudes et propositions (1995)
3. Guerfel, R., Sbaï, Z., Ayed, R.B.: On service composition in cloud computing: a survey and an ongoing architecture. In: IEEE 6th International Conference on Cloud Computing Technology and Science, CloudCom 2014, Singapore, 15–18 December 2014, pp. 875–880 (2014)
4. Guerfel, R., Sbaï, Z., Ayed, R.B.: Towards a system for cloud service discovery and composition based on ontology. In: Núñez, M., Nguyen, N.T., Camacho, D., Trawiński, B. (eds.) ICCCI 2015. LNCS (LNAI), Part II, vol. 9330, pp. 34–43. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24306-1_4
5. Han, R., Ghanem, M.M., Guo, L., Guo, Y., Osmond, M.: Enabling cost-aware and adaptive elasticity of multi-tier cloud applications. Future Gener. Comput. Syst. **32**, 82–98 (2014)
6. Herbst, N.R., Kounev, S., Reussner, R.H.: Elasticity in cloud computing: what it is, and what it is not. In: Proceedings of the 10th International Conference on Autonomic Computing (ICAC 13), pp. 23–27. USENIX, San Jose (2013)
7. Islam, S., Lee, K., Fekete, A., Liu, A.: How a consumer can measure elasticity for cloud platforms. In: Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering, ICPE 2012, pp. 85–96. ACM, New York (2012)
8. Jensen, K.: Coloured petri nets: a high level language for system design and analysis. In: Rozenberg, G. (ed.) ICATPN 1989. LNCS, vol. 483, pp. 342–416. Springer, Heidelberg (1991). https://doi.org/10.1007/3-540-53863-1_31
9. Liu, Z., Wang, S., Sun, Q., Zou, H., Yang, F.: Cost-aware cloud service request scheduling for saas providers. Comput. J. **57**(2), 291–301 (2014)
10. Naskos, A., Stachtiari, E., Gounaris, A., Katsaros, P., Tsoumakos, D., Konstantinou, I., Sioutas, S.: Cloud elasticity using probabilistic model checking. CoRR, abs/1405.4699 (2014)
11. Sbaï, Z., Barkaoui, K., Boucheneb, H.: Compatibility analysis of time open workflow nets. In: International Workshop on Petri Nets and Software Engineering (PNSE 2014), CEUR Workshop Proceedings, vol. 1160, pp. 249–268, June 2014. http://ceur-ws.org/Vol-1160/
12. Sbaï, Z., Guerfel, R.: CTL model checking of web services composition based on open workflow nets modeling. IJSSMET **7**(1), 27–42 (2016)

13. Inc Sun Microsystems.: Introduction to cloud computing architecture. Technical report, June 2009
14. CPN tool. http://cpntools.org/
15. Wu, L., Garg, S.K., Buyya, R.: SLA-based admission control for a software-as-a-service provider in cloud computing environments. J. Comput. Syst. Sci. **78**(5), 1280–1299 (2012)