Check for updates

Received: 7 February 2021 | Revised: 22 June 2021 | Accepted: 28 July 2021
DOI: 10.1002/cpe.6569

RESEARCH ARTICLE

WILEY

Designing a methodological framework for modeling and performance forecasting of self-adaptive cloud systems

Mehdi Sliem[1] | **Nabila Salmi**[1,2] | **Malika Ioualalen**[1]

[1]MOVEP Laboratory, USTHB, Algiers, Algeria
[2]LISTIC, Université de Savoie, Annecy le vieux, France

Correspondence
Mehdi Sliem, MOVEP Laboratory, USTHB, Algiers, Algeria.
Email: msliem@usthb.dz

Abstract

Autonomic computing got an increasing attention during past years in the context of Cloud computing in order to optimize the Cloud systems performance and increase their cost-effectiveness. In that direction, a broad range of various self-adaptability strategies have been proposed to overcome uncertainties related to Cloud systems like workload variations, hardware failures and malicious attacks. A special attention in the literature has been paid to resource allocation for cost and performance optimization, and self-healing for better faut-tolerance. However, while a lot of progress has been done on these topics, there is still a lack of a standardized methodology, easily reproducible and allowing validation and comparison of a self-adaptability strategies. This article proposes a methodology allowing to forecast the autonomic Cloud systems performances. The idea is to compare different self-adaptability approaches at the design stage, and identify the most adequate and optimized configuration. This methodology is based on a new architecture description language and a stochastic Petri nets-based modeling and simulation approach. We illustrate our methodology in this article through a running example and a set of experiments.

KEYWORDS

autonomic computing, cloud computing, performance modeling, resource allocation, self-healing, stochastic Petri nets

1 | INTRODUCTION

Autonomic computing is a principle which consists in enhancing a system with "Self-adaptability" features through an external feedback loop that tracks the system internal and environmental state, and operate the suitable changes based on the analysis of the monitored data. By this way, the analyzed system goals are consistently achieved and good performances are preserved with no human intervention.[1–3] These mechanisms are automated in an "autonomic manager" executing a loop, named MAPE-K loop as shown in Figure 1, this loop is based on an established knowledge that can evolve according to the system behavior. Traditionally, the analysis of the data monitored by an autonomic system is based on a combination of modeling, thresholds and forecasting methods to predict the future system state and define the most suitable system configuration to reach the predefined goals.[4–6]

One of the main research trends of autonomic computing application is self-adaptive Cloud systems.[7–14] Indeed, within large datacenters running some large-scale applications, the autonomicity of the underlying systems is a key feature to meet the performance, availability and service level agreements (SLAs) while ensuring a cost effectiveness of offered services. During our study, we identified two topics in the context of self-adaptive Cloud systems that got increasing attention over the last years: (a) self-adaptive resource allocation in an on-demand basis and (b) self-healing feature that provides fault-tolerance capabilities.

Optimizing the resource allocation in Cloud systems becomes a key challenge as different types of tasks or applications with varying workloads are requested from customers of a Cloud. The required number of computational resources needs to be available and assigned when the users

 © 2021 John Wiley & Sons, Ltd.

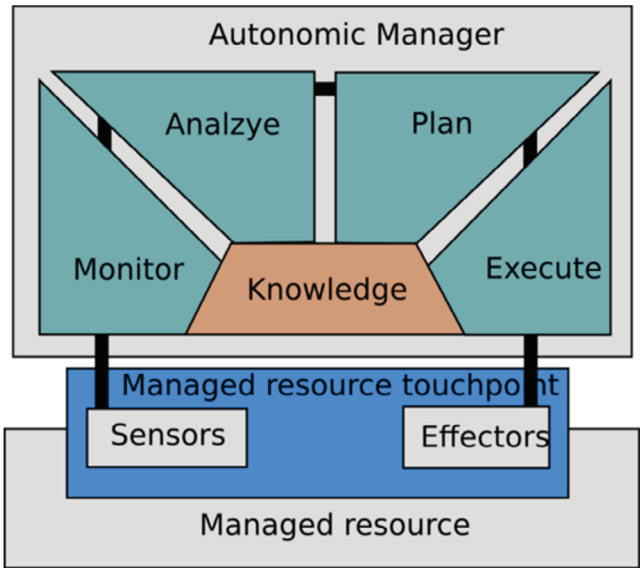

FIGURE 1 Autonomic loop

workload increases. This requires a performant forecasting strategy to scale the system at time and meet the customer SLA. On the other hand, the system must not be over dimensioned which would lead to an increase of costs and power consumption. In that direction, autonomic computing appeared to be a suitable concept to cope with the workloads uncertainty and several self-adaptive strategies have been proposed in the literature in order to provide Cloud infrastructure with efficient resources management.[15–32] Moreover, some other work focused on the continuous availability of the resources, which is achieved by self-healing properties implemented by the disruption detection, the diagnostic of failure root causes, deriving a remedy and recovering with a sound strategy.[33–41]

However, although a lot of progress has been made toward performant self-adaptive Cloud systems, there are still some gaps that need to be addressed.[42] is one of the rare surveys that specifically examines methodologies in self-adaptive system research: the authors analyzed the most recent primary researches in term of self-adaptive strategies. Among the major findings of that survey, the following are of high interest, in our opinion, in the context of Cloud resources management. First, the repeatability of the experimental results is generally poor which makes it difficult to compare two different self-adaptive strategy. Secondly, the authors pointed out that there is a very strong bias in the field toward positive results. Besides,[43] attempt to overcome the lack of standardized validation methodology in the context of self-protection and cybersecurity of autonomic systems by introducing a methodological framework for performance evaluation, but we found no similar study in the context of self-adaptive resource allocation and self-healing in Cloud systems. Additionally,[44,45] highlighted need of self-adaptive systems to handle complexes types of uncertainties and there is no general forecasting method that performs best in all scenarios.

These problems therefore motivated us to direct the current study. This article presents a generic analysis and modeling methodology allowing to forecast performances of autonomic Cloud systems, by comparing different self-adaptive resource allocation and self-healing strategies at design time. Our modeling and simulation approach is based on stochastic Petri nets-based (SPN). The SPN is a strong mathematical model, widely used for complex systems performance analysis, which makes the experimental tests easily repeatable.

For a concern of genericity at design time, we also introduce a new architecture description language (ADL), able to describe a cloud system with self-adaptive capabilities. We also developed a tool automating analysis of a system or strategy basing on its description using the ADL. The tool identifies the best self-adaptive configuration or strategy to apply on the system under study. Our methodology is presented in this article through a typical example of self-adaptive Cloud system to illustrate our approach.

2 | THE RUNNING EXAMPLE: A MULTI-TIER CLOUD APPLICATION

To illustrate our approach along this article, we use a typical example of a public application running on a Cloud infrastructure. The application is hosted on several tiers running the different services required to process a user request. Each tier run on different servers with a replication policy and each server has a resource set to be allocated to user requests such as virtual machines (VMs), memory space, processor cycle or network bandwidth. We assume that these requests are organized in service classes, where a service class indicates the resources amount required to process requests of the corresponding services. Finally, each tier has a dedicated autonomic manager.

User requests arrive through a firewall. They are first analyzed according to some predefined security settings, to prevent attacks, like denial of service (DoS). Authentication and digital signature are performed. As soon as we detect a suspicious request, a handler is invoked.

In our Cloud hosted application, the tier VMs are load balanced. This allows, in one side, to fault-tolerance within a VM or domain failure, and in the other side, prevents contention situations and bottlenecks. Once a request is processed by a tier, it requires either a call to the next level, or it generates a return to the customer or the previous tier.

A continuous monitoring is performed: a reliability service periodically analyzes the resource group state to check failure occurrence and scale the resources toward an optimal state. At the same time, a scalability service allows VMs to increase their resources to face higher workload or to auto-scale down while workload intensity is low.

In this work, for a given server three state levels are considered:

i. A stable state where the server operates correctly,
ii. A degraded state that affects the server performances by decreasing the efficiency of resources use.
iii. A failure state, where new requests can no longer be processed and the server can no longer find its stable state autonomously.

Degradations may be due to non-detected attacks or software/hardware failures. When the autonomic loop detects and diagnoses a degradation, a recovery plan is prepared and the server is reset to a stable state. But a human intervention is obviously necessary to fixe hardware failure.

3 | MODELING FORMALISM: STOCHASTIC PETRI NETS

3.1 | Definition

Complex system analysis is widely done using the stochastic Petri nets (SPN)[46–51] that we use for our modeling. The choice of this formalism is first motivated by the need to a state-based model that has the ability to evaluate quantitative performance indices related to system configurations. This is in contrast to other performance formalisms like Queuing networks or process algebras models where conflicts cannot be modeled. Moreover, as nowadays systems are composed from several interacting components, and as our methodology is incremental, we need to compose sub-models to connect multi-tier sub-models like VMs groups sub-models. In this direction, Petri Nets are compositional models: interaction between Petri net models representing interacting sub-models, may be easily defined through merging transitions/places. So, we define formally an SPN by the following.

Definition 1. (Stochastic Petri Net). A stochastic Petri net is a couple $N = (\grave{N}; 0)$ defined with:

- $\grave{N} = (P, T, Pre, Post, Inh, Pri)$ is a Petri Net where P is a set of places, T a set of timed transitions with a stochastic firing delay. *Pre*, *Post*, *Inh* are respectively precondition, postcondition and inhibition functions relating transitions to places, and *Pri* the transition priority function.
- $\theta : T \times Bag(P) \rightarrow \mathbb{R}^+$ where $\theta(t, M)$ is the firing rate of t in M, that is, the parameter of its exponential law, where $\theta(t)$ represents:
- The weight of t if $Pri(i) > 0$ (t is said an immediate transition).
- The firing rate of t if $Pri(i) = 0$ (t is timed): the enabling duration before the firing of $t(c, M)$ follows an exponential probability distribution with mean $\theta(t)$.

A state of an SPN, called "marking", is a vector giving for each place the number of contained marks.

3.2 | Performance analysis of an SPN

The quantitative analysis of an SPN model is possible thanks to their equivalence with Markovian processes, which are well-known for their numerical results. It is conducted through several steps:

1. The state space (said the reachability graph) of the SPN is first generated.
2. Then, a set of qualitative properties are checked, namely liveness, boundness and the existence of an initial state, to ensure an ergodic Markovian chain and the existence of a steady state.
3. The steady state probability vector is computed by solving a system of equations.
4. Finally, performance parameters are computed using the obtained steady state probability vector, such as the average transition firing frequency, the average place marking, the probability of a given event and the average sojourn time in a place. At a practical level, these parameters represent mainly response time of a task, throughput or specific performance indices.

Multiple tools exist for Petri nets editing and quantitative analysis.[52] compares twenty Petri net tools in different aspects. Among those, the GreatSPN tool[53] is the main one supporting the stochastic Petri nets, the timed transitions and the structural and advanced performance analysis. It is therefore this tool that has been chosen for our experiments.

4 | MODELING APPROACH

To define a generic modeling approach useful in most Cloud configurations, we first present the key concepts for the autonomicity of a tier. Then, we follow a defined methodology, to build a reliable model allowing accurate analysis results.

4.1 | Key concepts

In this section, we identify a key concept set that can possibly affect performances of autonomic multi-tiers systems. These concepts should be considered when applying our methodology on a given system.

4.1.1 | System's resources

The first element to model in any system is a system resource, as it is the main element managed and shared between users. In our modeling, resources, their states, their allocation and their distribution over the time have to be represented. The server state with a resource distribution represents the system configuration. For example, in a saturation state, the system becomes unable to process new requests because all available resources are consumed by current requests. In a non-optimal state a required QoS cannot be satisfied. Modeling the resource is then a key point to analyze the system capability to maintain a normal state with an optimal resource usage.

4.1.2 | Failures

Hacker attacks, software errors and hardware breakdowns are considered as main faults treated by self-healing in a cloud system. Those faults are prevented by implementing several security mechanisms. System performances, reliability and trustworthiness are directly influenced by the degradation of a component. Modeling failures and their frequencies allows predicting their effect on performances and identifying their consequences when several degradations appear at the same time. In our case, we do not focus on a specific failure, but model an abstracted view of it.

Whenever the designer is interested in a defined failure, the cause and evolution of this failure are to be modeled, to highlight its impact on the system. This is done by a modeling expert.

4.1.3 | Security strategy and fault-tolerance

Important mechanisms to be modeled are those consuming lot of resources affecting then the performances of the system. Particularly, security mechanisms such as load balancers, replication, firewall behavior have to appear in the model, so that to make system performances prediction more realistic, including crucial factors affecting performance. In addition, modeling these features becomes necessary during fault-tolerance implementation, to evaluate the system reliability and its autonomy level to overcome component failures.

In the case of attacks and failures, a specific fault-tolerance technique or a security strategy is triggered, we assume, a given strategy for each situation.

4.1.4 | Autonomic loop

For system reliability, a recovery plan and some security mechanisms should be predefined. Comparing drawbacks and efficiency of recovery strategies implemented in the autonomic loop, for a system configuration, is driven by modeling the system configuration itself. Hence, the model should represent both of the system and its autonomic loop. This allows to measure, at the same time, the system configuration performances and the autonomic features impact on the system. Such model makes possible to estimate the autonomic loop influence for different system configurations. Moreover, the system state and its evolution over time are directly visible on the model.

4.2 | Overview of our methodology

To model an autonomic cloud system, we propose an incremental methodology based on modeling key concepts given above. We present in this section our modeling methodology as shown in Figure 2:

- We first start from a description of the studied system. The description may be given in any design language, such as UML (Unified Modeling Language), SysML (Systems Modeling Language) or an appropriate Architecture Description Language (ADL). Such description must include interaction between system components and the autonomic loop specifying reconfigurations operated by the autonomic manager on a tier.
- Then, each tier is studied in isolation and modeled separately according to its characteristics: number of servers, number of available resources, number of resources required to allocate at once and the implemented autonomic features. The tier modeling is done in an incremental manner, as follows:
  - The requests arrival is first modeled, including workload intensity and pending requests queue.
  - The requests analysis is then represented in the same way, such as the manager part invoked in case of suspicious arrivals.
  - VMs of the same tier are modeled separately. Each VM model covers the server resources, their allocation and their use for processing requests.
  - Then, we represent, in the model, each VM state and its relationships and influences on the server behavior.
  - After constructing the different tier components sub-models, these latter are assembled by merging specific transitions modeling component communication parts. This gives the final model of each tier of the system.
  - Finally, all autonomic loop steps implementing self-healing properties must be modeled. Additionally, the system autonomy level is evaluated by modeling the required human intervention that fixes serious failure.

FIGURE 2 Modeling methodology

- Once all tiers are modeled, we assemble the obtained parts, according to their interactions. We model two kinds of interactions between a current tier and its neighbors; either a request processing needs a submission to the next tier, or the processing is finished, letting response being sent to the previous tier or client. This step finalizes the model construction and gives the global model of the whole system to analyze.
- Finally, from a specification of the targeted performances indices, a performance analysis is performed on the system global model to quantify the system performance and its quality of service (QoS).

The modeling methodology explained below highlights the general skeleton of a multi-tiers autonomic system model architecture. Each part is replaced with the appropriate sub-model according to the real system to assess.

4.3 | Tiers modeling

Following our previously proposed methodology steps, we give obtained sub-models of the different parts of a multi-tiers autonomic cloud system.

4.3.1 | The request submitting and analysis sub-model

Requests is the name of the place modeling a request. The number of tokens marking this place is given by the number of user requests. The arrival and queuing of a request in the system are modeled by the transition *NewReq* whose firing rate represents that of the request arrival (see Figure 3).

The system processes the received request by first analyzing it if it is suspicious or not (transition *ReqAnalyze*), according to a predefined strategy. This analysis models a firewall or an IDS (intrusion detection system) processing. The firing rate of the transition *ReqAnalyze* depends on the restriction of the strategy. The *RequestAnalyzed* place represents analyzed requests. If an attack is detected (transition *SuspiciousArrival*), a manager is triggered (transition *Handler*) to process this kind of request. Otherwise, the request is sent to the first tier's queue to be processed.

Note that a token (representing a request) is put back into the *Requests* place, once processed, to model the arrival of a new request and keep the whole model liveness.

4.3.2 | VM sub-model

The VM model (Figure 4) highlights two important entities: the VM's queue (modeled by the *QueueS1* place) and the VM's resources (*ResourcesS1* place). The *AllocatingResourcesS1* transition models allocation of a set of resources to each request. Once the allocation done, the user request is processed through the transition *ProcessingS1* with the corresponding firing rate. Finally, if the server is in a degraded state, another transition modeling processing (P*rocessingS1Deg*) is added with a lower rate.

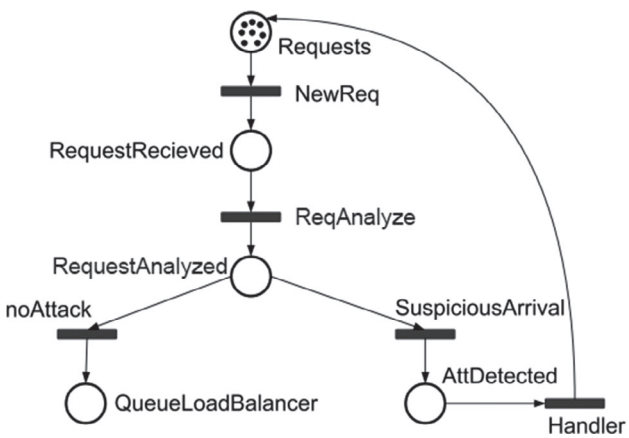

FIGURE 3 The request submitting sub-model

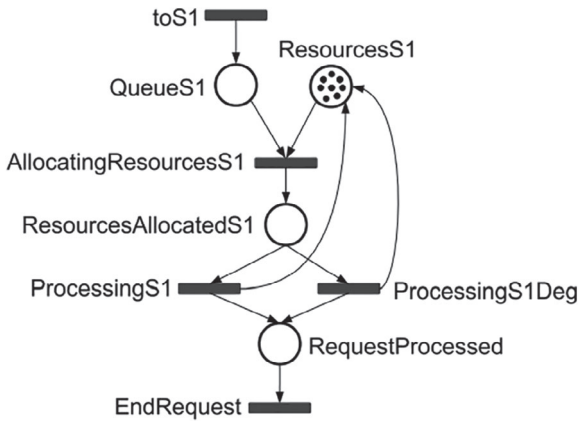

FIGURE 4 The VM sub-model

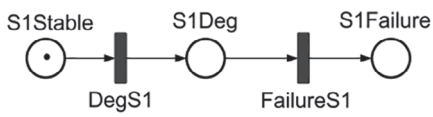

FIGURE 5 The VM state sub-model

4.3.3 | Server state sub-model

A VM server is first in a stable state, then passes to a degraded state in case a problem occurs, and finally reaches a failure state. Each of these states are represented in Figure 5 by a place (*S1Stable*, *S1Deg*, *S1Failure*). The transition, from a state to another, is modeled by the transitions *DegS1* and *FailureS1*; Inhibitor arcs are then connected to the processing transitions to model the effects of the server state on its behavior. The modeling of the effects of the server state on its behavior is done by connecting inhibitor arcs to the processing transition. Inhibitors are connected from the *S1Off* place to processing transitions. The state switching from a state to state is modeled in the autonomic sub-model, with transitions connected to state places according to the associated plan.

4.3.4 | Autonomic loop sub-model

- The autonomic loop monitoring phase is modeled by the first part: launching one autonomic loop at a time is represented by a place named *Monitoringallowed*; to start the monitoring, a transition *startMon* is triggered. After that, each server state is analyzed S_i*DegDetected* and *noFailure*S_i through the S_i*Analyze* places, where S_i refers to the server name. If the analysis does not detect any issue, the *noFailure*S_i transition allows the monitoring to move toward the next server analysis.
- The transition S_i*DegDetected* detects the degradation when a token is present in the S_i*Deg* place. The tokens of both the S_i*Deg* and S_i*Analyze* places are consumed, modeling the diagnosis of the server state. The *Plan*S_i place prepares the act part, which chooses a solution and recovers the stable state of the server by applying the solution. This is modeled through the *Recovery*S_i which puts the token back to the S_i*Stable* place. The second server analysis (place S_{i+1}*Analyze)* can be driven after that, to continue the monitoring.
- Finally, we model human intervention when automatic solution is not possible: the S_i*Failure* place represents servers requiring human intervention, where the *HumanIntS*$_i$ transition models the human operation, putting back the server to the stable state (token returned to the place S_i*Stable)*. All the tier will be unavailable, when the two servers are out of service. In this case, the unresolved failure is fixed by the last transition *HumainInt*.

To model the whole autonomic multi-tier system, the presented sub-models are gathered as shown in Figure 6.

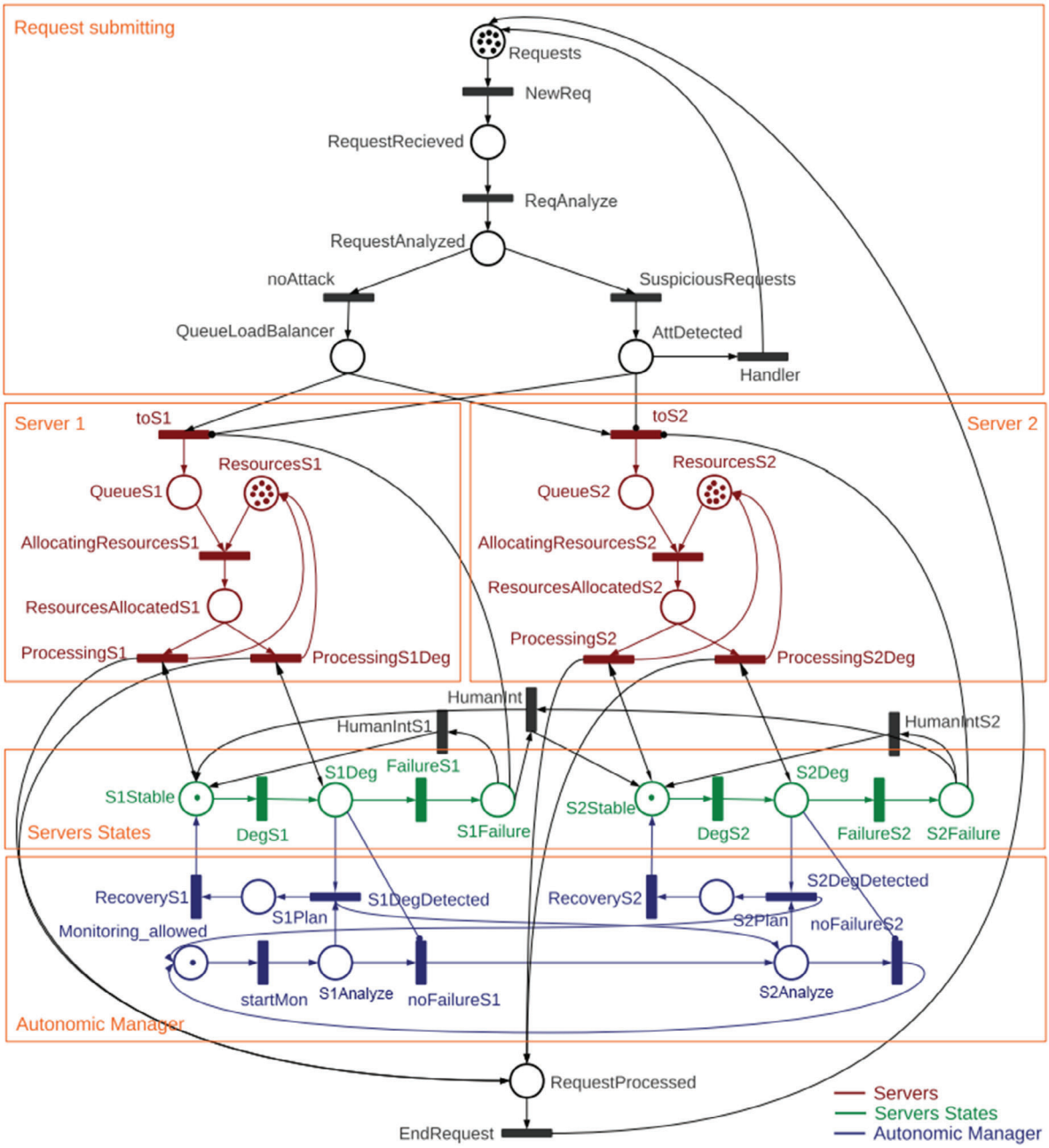

FIGURE 6 Global system model

4.4 | Connecting two tiers sub-models

Two tiers are connected through the *NextTier* and *BackToClient* transitions: the first transition models the case where the request is forwarded to the next tier, and the transition is then linked to the place *Queue* of the next tier. Otherwise, the request ends, putting back the corresponding token into the previous *Requests* place (see Figure 7).

5 | ANALYSIS OF A MULTI-TIER SYSTEM AND SCALABILITY

Our modeling methodology produces a model that can be used for analyzing a one-tier system. However, when dealing with several tiers, like in modern internet applications, the obtained model becomes rapidly important, and generates a great number of states, making difficult and even

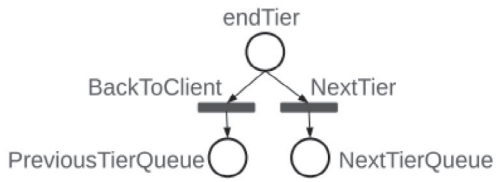

FIGURE 7 Connecting the sub-models of two adjacent tiers

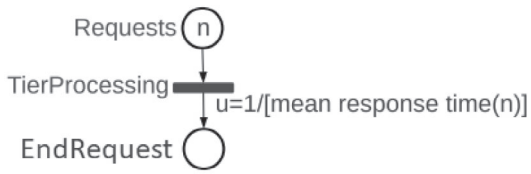

FIGURE 8 Reduced model

impossible model analysis. The growing scalability of obtained models is then a main issue to consider in order to enable its analysis, using our approach, in a reasonable time. To this purpose, we proposed the use of a Petri nets reduction technique to keep a reasonable time when analyzing a system models.

5.1 | Principle

Our Petri nets reduction technique consists, for analyzing a tier i, in substituting SPN models representing other tiers $j, j \neq i$, with a simple equivalent model, called an "abstract view" modeling the concerned process. For instance, to analyze a 2-tier system, the second tier is first analyzed as a single tier. Then, for analysis of the 2-tiers model, the second tier is replaced by its "abstract view". By the way, for analyzing a 3-tiers system model, the second and third tiers are each replaced by their respective "abstract views". Let us formalize and detail this technique.

Let S be the autonomic system under analysis, being composed of n tiers T_i, $i = 1, \ldots, N$, to which models M_i, $i = 1, \ldots, N$ are built. To analyze this system, we propose to begin with the last tier and conduct analysis gradually through several steps: each step studies a tier model M_i, connected to "abstract views" of other models M_j, $j \neq i$, rather than connecting the whole tier models, getting an important model. Let us define, first, what is an "abstract view" of a tier.

5.2 | Abstract view of a tier

The *abstract view* of a tier is a macro representation or *macro-view* that aggregates its behavior to a minimal equivalent Petri net keeping interactions with other tiers.

Definition 2. (Abstract view of a tier). *Let M_i be the SPN model of a tier T_i. A(Mi) is the abstract view of M_i where: A(Mi) = (P, T, u)* Ǹ = (P, T, *Pre*, Post, Inh, *Pri*) *is a Petri Net where P = {Requests, EndRequest}, T = {TierProcessing}with a firing delay u.*

The abstract view A(Mi) of the tier T_i is composed of:

i. *A timed transition TierProcessing, modeling the tier activities. This transition summarizes all the Petri net connected between the places Requests and EndRequest in* Figure 6,
ii. *An input place Requests, modeling the workload received by the tier and marked with a number n of requests; this place corresponds to the Requests place in* Figure 6, *and*
iii. *Output place EndRequest, representing the end request processing of the tier, corresponding to the EndRequest place of* Figure 6.

To the transition is associated a firing rate u inversely proportional to the response time of the n requests marking the input place, which means: u(TierProcessing, M) = 1 ÷ ResponseTime(M).

Figure 8 shows the abstract view model.

5.3 | Analysis process

The analysis process proceeds as follows:

- First, we build the model of each tier T_i. Let us denote the obtained model M_i.
- We isolate the model M_N of the last tier. A complete analysis of this model is, then, performed to compute a mean value of the response time of the tier under different initial markings and transitions rates.
- We move to do then the analysis of the model $M_N - 1$ of the tier T_{N-1}. To do that:
  - We isolate the model M_{N-1} of the tier $N - 1$, beginning from the Requests place of the tier, as if it is the 1st tier.
  - The analyzed tier's model M_N is replaced by its abstract view $A(M_N)$.
  - The abstract model $A(M_N)$ is connected to the model M_{N-1}, according to the system specification and following our methodology proposed in 7.1. Figure 7 shows an example of an SPN modeling both cases of client answer return and request forwarding between the two tiers. This step gives a new aggregated model, denoted $AGGR_{N-1}$ representing the two tiers T_N and T_{N-1}. The analysis of this aggregated model will result in a reduced number of states.
  - We analyze the obtained model $AGGR_N - 1$ and compute the mean response time of the tier T_{N-1} under different initial markings and transitions rates.
- We move to the analysis of the model M_{N-2} of the tier T_{N-2}. The reduction is applied again on the model $AGGR_{N-1}$, reducing the two last tiers combined within a new abstract model $A(M_{N-1})$. We repeat the construction step of the aggregated model of the tier T_{N-2}, then the "analysis" step.
- The previous steps are repeated until all tiers are considered. The recursive reduction of each analyzed part of the system ensures to keep a reasonable state space of the obtained Petri net, regardless of the total number of tiers in the system. The final aggregated model analysis (of $AGGR_1$) gives then performance and reliability metrics of the whole system, avoiding a combinatorial explosion of the state space number.

Figure 9 summarizes our reduction method. This technique has as benefit to significantly reduce the state space of the SPN. Therefore, this analysis technique allows to ensure the scalability of our methodology and to get faithful results according to the unreduced model.

In the next section, we will show how to automatically apply our methodology by introducing a new architecture description language. The SPN model will then be automatically generated by our tool starting from a description of the system to analyze.

6 | CLOUD ARCHITECTURE DESCRIPTION LANGUAGE AND MODEL GENERATOR

For automating our modeling according to different autonomic cloud systems, we propose a general component model allowing to describe this kind of systems, provided with necessary information enabling automatic component performance modeling. We also introduce an Architecture Description Language allowing to describe cloud components properties as well as self-adaptive components, system's resources and environment and performance characteristics. These specifications are used to derive, from architecture descriptions and component behaviors, an automatic mapping into performance SPN models, without additional manual modeling.

6.1 | Cloud ADL model

We present now the main components considered in our Cloud ADL based on a textual XML syntax:

- Physical resources: This concept helps to define basic resources usually held by a cloud system as physical machines and processors cores frequencies.
- Virtual resources: describe the structure of virtual machines created by the system over the application execution and their mapping to physical resources.
- Classes of service: allow to present different services offered by an application and their processing time.
- Workload intensity and variations: these notions define a test scenario for the generated model, through defining possible intensities of user workload, switching times and duration for each state.
- Autonomic manager: describe monitoring thresholds and automatic reconfigurations for resources scaling.

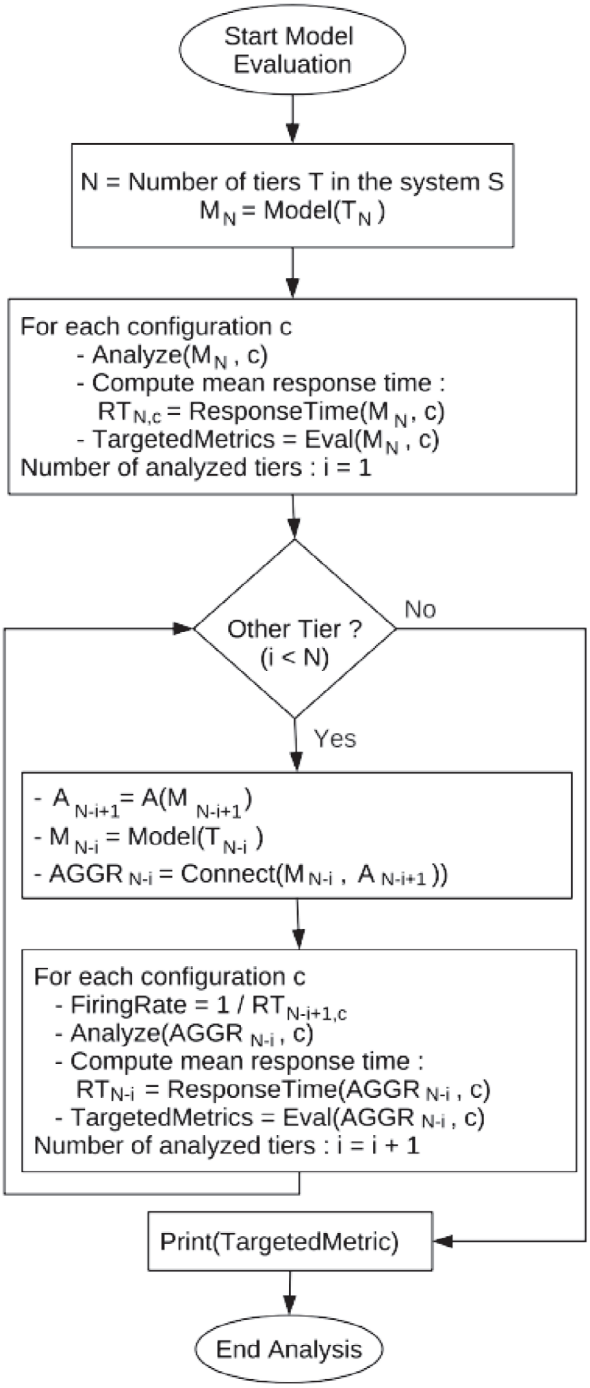

FIGURE 9 Analysis using reduction method

Figure 10 gives the meta model of our ADL. Based on this ADL, we developed a tool called "Autonomica", implementing our modeling methodology, and generating automatically the formal SPN model from a system description (Figure 11). The resulting file describing the SPN can then be used with a Petri Net analysis tool to compute the targeted performance metrics.

The next section will show a SPN model generated by the Autonomica tool.

6.2 | A self-adaptive resource allocation cloud platform

We present, in this section, a typical example of resource allocation in an autonomic cloud system.

This system contains an autonomic cloud server receiving user requests and distributing a set of resources to these requests through a resource allocation process. Requests in this example are served following a FIFO policy, so conflicting requests are served according to their arrival order.

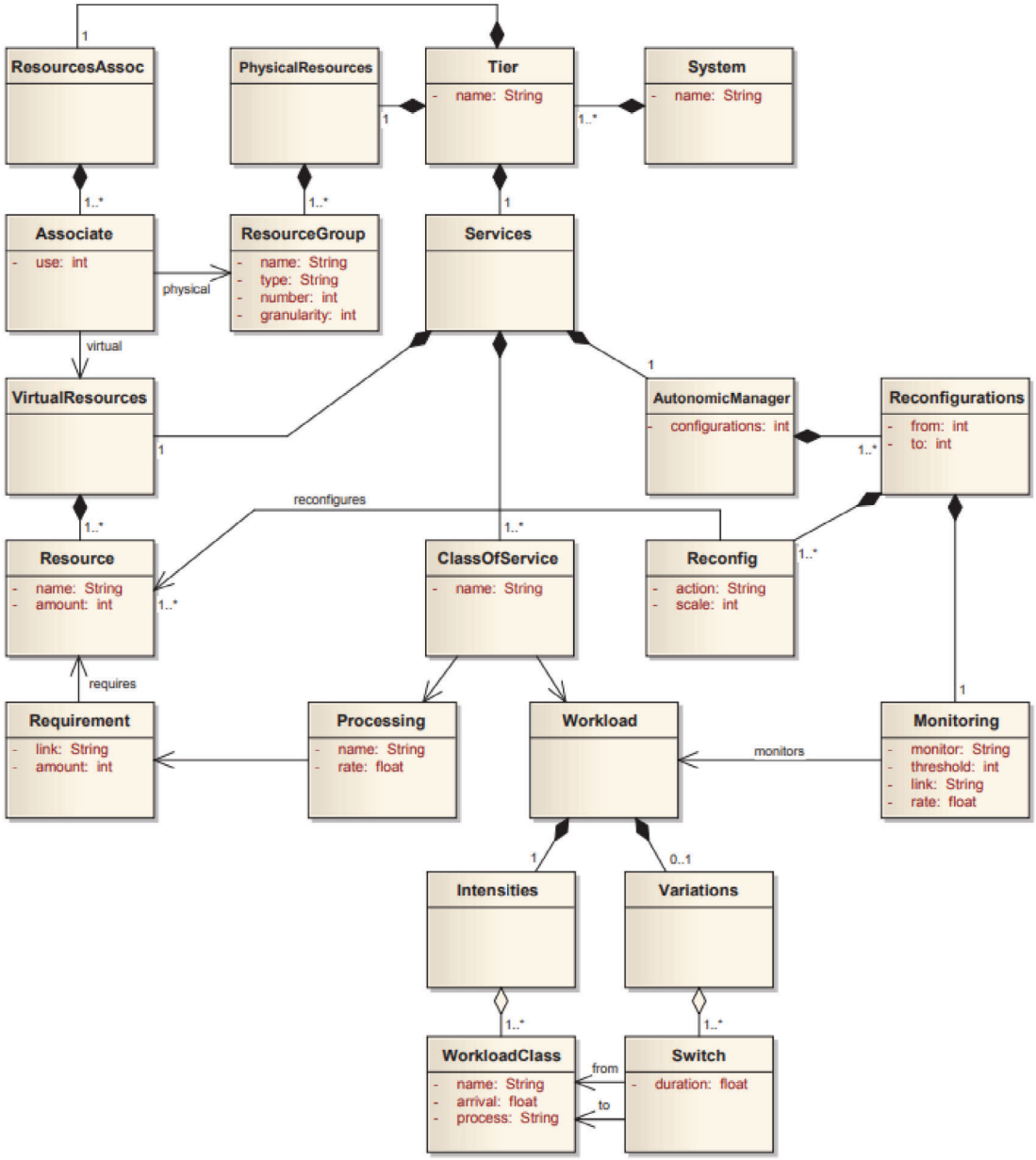

FIGURE 10 Architecture description language

The autonomic handler performs a scaling up/down strategy, to increase/decrease the system scalability according to the workload. This process is carried out by, adding/removing resources from the resources pool.

6.2.1 | Generated model

Using the associated component description of our example, the Autonomica tool generates the SPN of Figure 12, based on the proposed methodology. The obtained model can then be tested, using different initial marking and firing rates, to identify the best autonomic manager configuration. Identifying the best autonomic manager configuration is done by testing the obtained model, using different firing rates and initial marking.

7 | ILLUSTRATION

In this section, a set of experiments is directed on our running example. To more illustrate our performance analysis approach and show its efficiency, we study variations of the same system with modified properties.

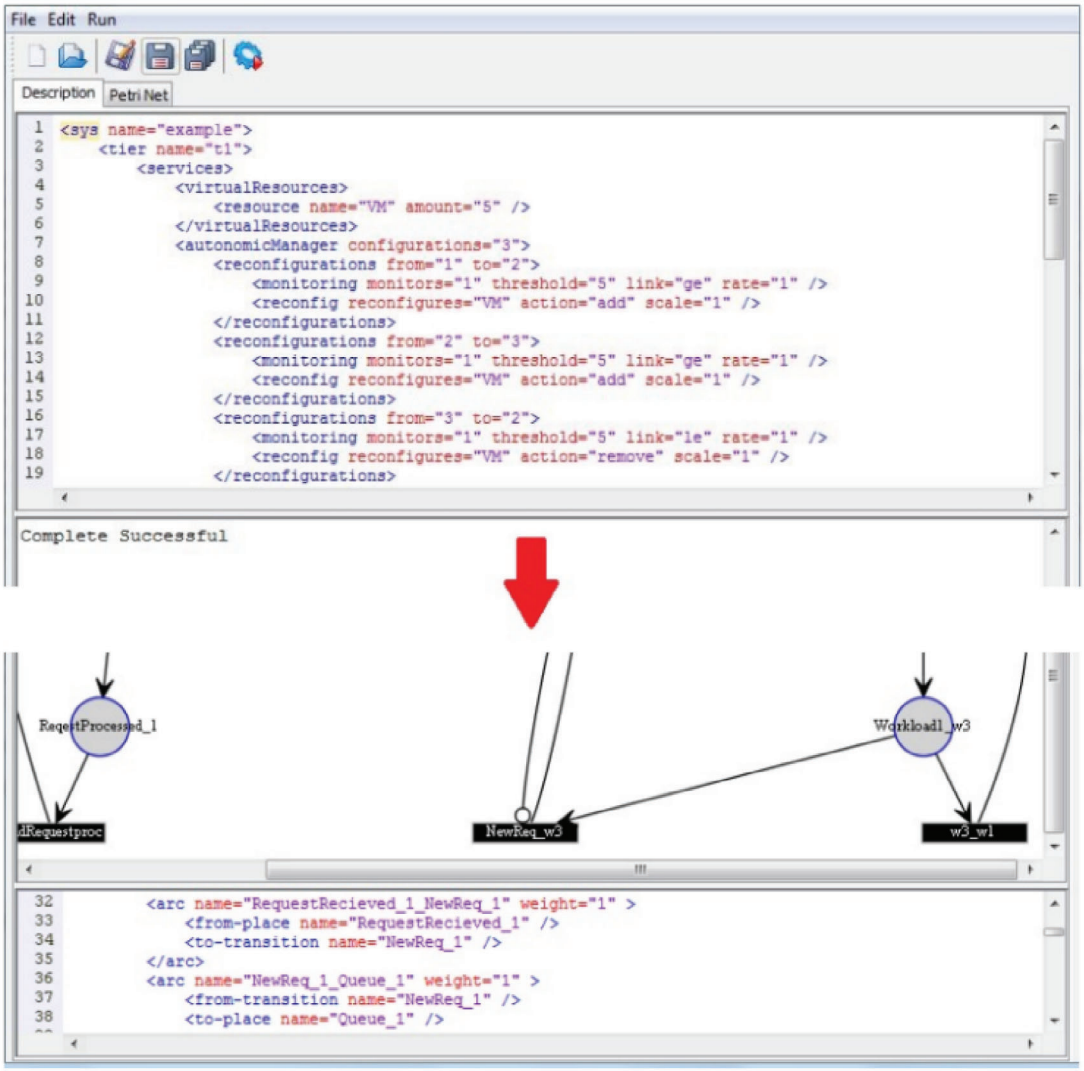

FIGURE 11 User interface of the Autonomica tool

7.1 | Systems under test

Let us consider four configurations of our autonomic cloud system example:

1. System 1: We first present the experimental results on the presented running example modeled by Figure 6. The system is fault tolerant with a load balancing between its servers. Each server is, continuously, monitored. A recovery plan is executed when the server is in a degraded state, negatively affecting the performances. Human intervention is inevitable to fix the problem in the case of important failure on one server.
2. System 2: In a second experiment, we introduced resource consumption in the autonomic process of the previous system. Consequently, to fix failures, the autonomic loop consumes resources when performing the recovery plan, possibly affecting system performances.
3. System 3: The third experiment considers the running example, but using a different reliability strategy: In this system, the reliability strategy uses servers as replica that are initially turned off. When the autonomic manager detects a failure state on a server, another one is turned on in order to maintain system availability. The replication strategy is, here, the basis of self-healing by providing some servers when the main ones are out of service. This strategy is therefore focused on high availability.
4. System 4: Finally, we address the case of the system described in Section 7.2, considering a system with resource
5. Allocation strategies, generated by our tool.

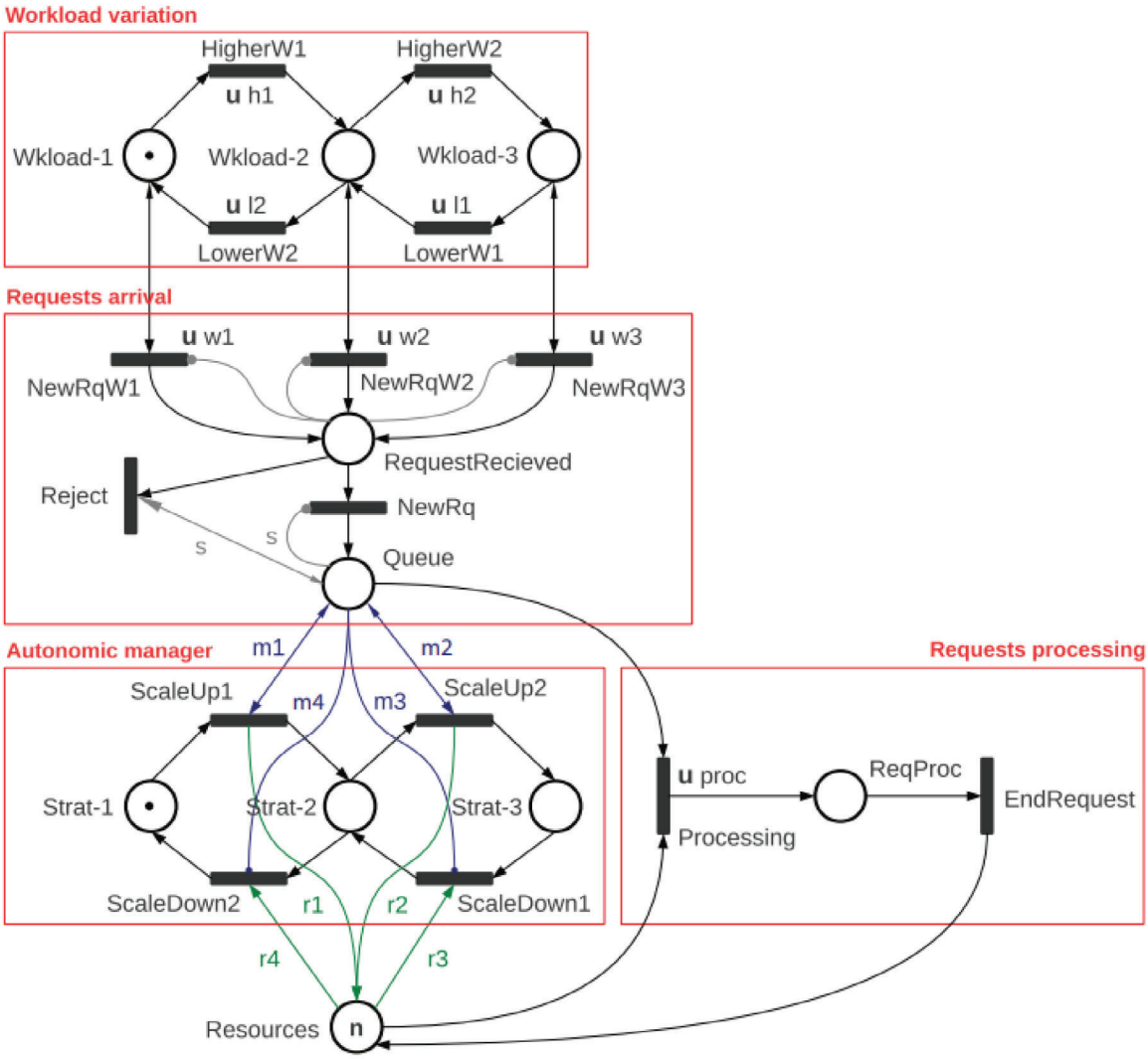

FIGURE 12 Generated Petri net

7.2 | Experimental results

We analyzed, in our experiments, the systems described above, under various configurations. The obtained models were analyzed using the GreatSPN package [,[4752]], on an Ubuntu 12.4 workstation with 4 GO of RAM. Tests are automated under several configurations by using Perl scripting.

7.2.1 | Experimentation parameters

We used several configurations for each studied system, by varying the initial markings of models. A marking is given by the following values: requests number, number of available resources, failure frequency, processing rate in degraded state, periodicity of the autonomic loop and number of suspicious requests.

The system workload intensity is defined by combining the initial marking of requests number with the rate of requests arrival, while the resources number for each server allows to test different configurations. The number of requests, that trigger the firewall manager or require a special process, is represented by the rate of suspicious arrival.

The probability of a failure occurrence, within a given server per unit time period, defines the failure rate. In our tests, failure rates of server instances are assumed to be equal.

Finally, the autonomic loop rate defines the periodicity of monitoring and analysis operated by the autonomic handler. A lower periodicity implies. That the autonomic loop is more frequently executed. In our tests, we specified different values of periodicity.

7.2.2 | Performance and reliability study

We study the evolution of several metrics from obtained steady-state probabilities. The configuration efficiency is evaluated by considering the following metrics of reliability and performances:

- Throughput of processed requests.
- Mean response time for a user request.
- Resources consumption and system availability.
- Necessary human intervention when the whole system is out of service.
- Mean time between failures.
- Frequency of autonomic recovery.
- Reliability coefficient.

7.2.3 | Testing "System 1" example

We start by computing the targeted metrics for our first illustration example. Figure 13 depicts the throughput of processed requests for the first strategy (Figure 6) under different processing rates in a degraded state (*ProcessingSiDeg* transition): the throughput increases constantly with loop periodicity increasing, following a logarithmic evolution. Because the loop in our strategy does not consume the resources of the server. The functioning of the system is improved by the autonomic intervention. Contrariwise, the increase of failure frequency decreases the throughput.

In the same direction, the mean response time, shown in Figure 14, is improved with more autonomic loop intervention, but increases with higher failure rates due to a longer processing time when a server is in a degraded state. The total failure, contrariwise, does not affect the response time, but affects the throughput and frequency of processed requests.

Figure 15. shows how many times the whole system will be out of service because of failure of all tier servers what requires human interventions. This metric provides information on the evaluated system reliability and its level of autonomy and fault-tolerance. The necessary intervention number increase with a higher failure rate and decrease with more frequent autonomic loop executions. Expected.

Then, we evaluated the reliability level through some metrics (Figure 16): in top figure, depicts the variation of the mean time between failures (MTBF) of a server, according to both of the failure rate and the loop periodicity. The middle figure shows the autonomic recovery frequency without downtime. This frequency start by increasing when the failure rate increase then decreases from a certain failure rate threshold, because important failure cannot be repaired by the autonomic loop. The reliability coefficient, depicted in bottom figure, improves slightly when the loop periodicity increases and is inversely proportional to failure rate.

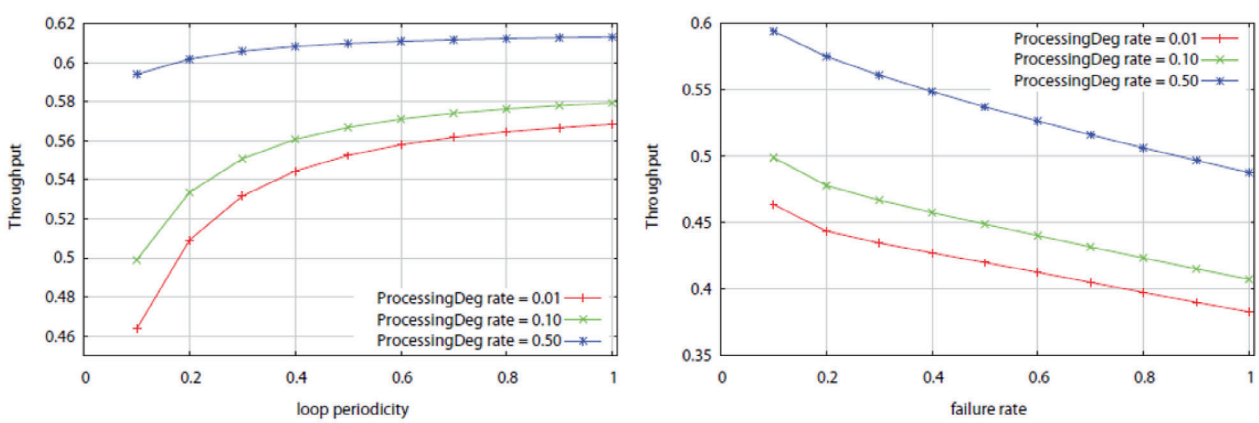

FIGURE 13 Throughput of processed requests

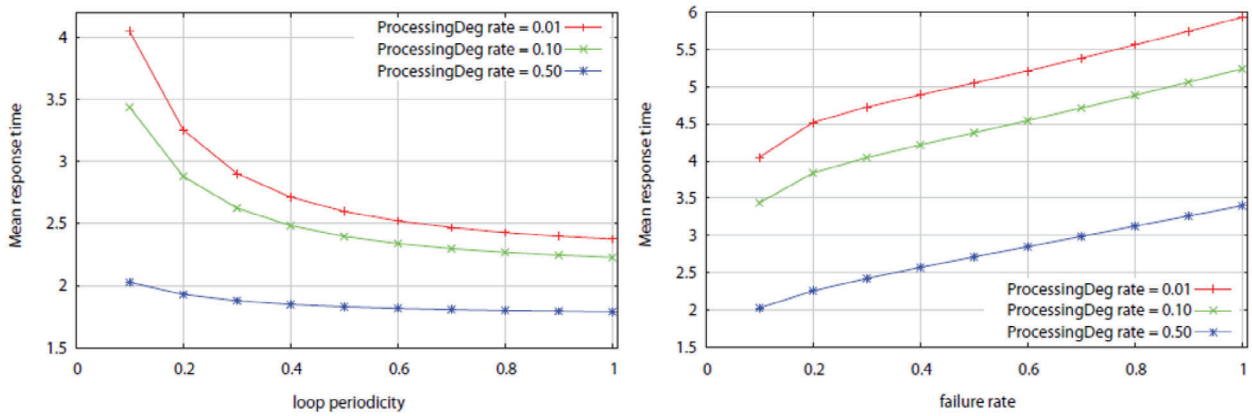

FIGURE 14 Mean response time

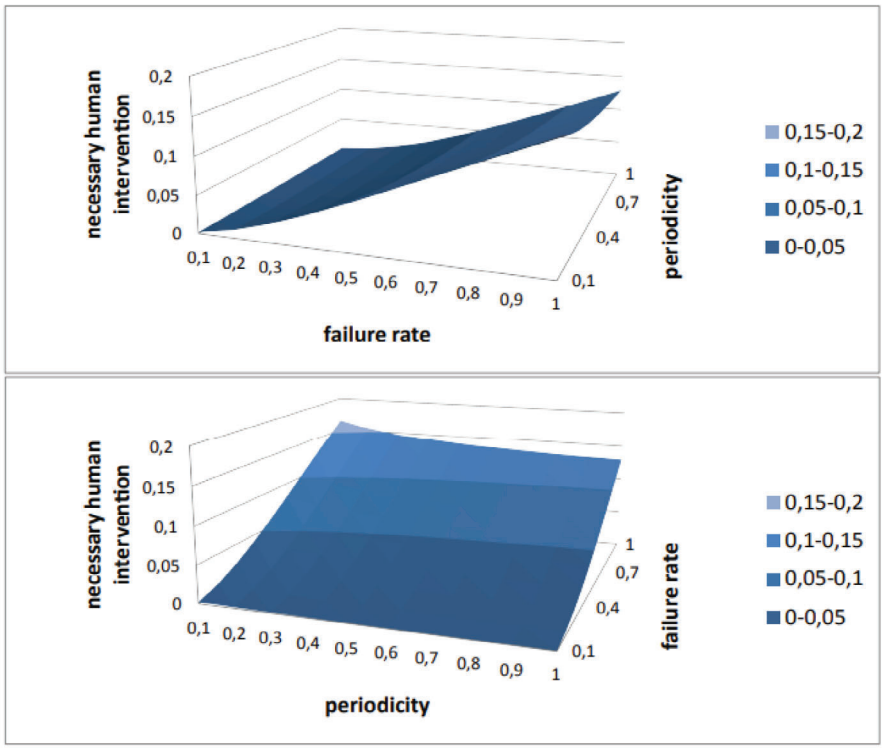

FIGURE 15 Necessary human intervention

7.2.4 | Testing "System 2" example

The metrics, presented previously, follow the same evolution according to the autonomic loop resources consumption and depends in the same way on the variation of the various parameters. However, general performances are, for some configurations, negatively affected by the periodicity of the autonomic loop.

The left of Figure 17 shows how the throughput of requests processing increases with a unique resource, a processing rate of 0.01 in degraded state (ProcessingSiDeg) and a failure frequency of 0.1. We notice the same behavior on the right of the figure at the beginning, where the tests have been performed for three different performances on a degradation state (rate of 0.01, 0.1, and 0.5 for the ProcessingSiDeg transition), all with two resources consumption in the model. We notice then that above a relative threshold, the loop periodicity decreases the throughput of requests processing because of the resource consuming of the autonomic manager.

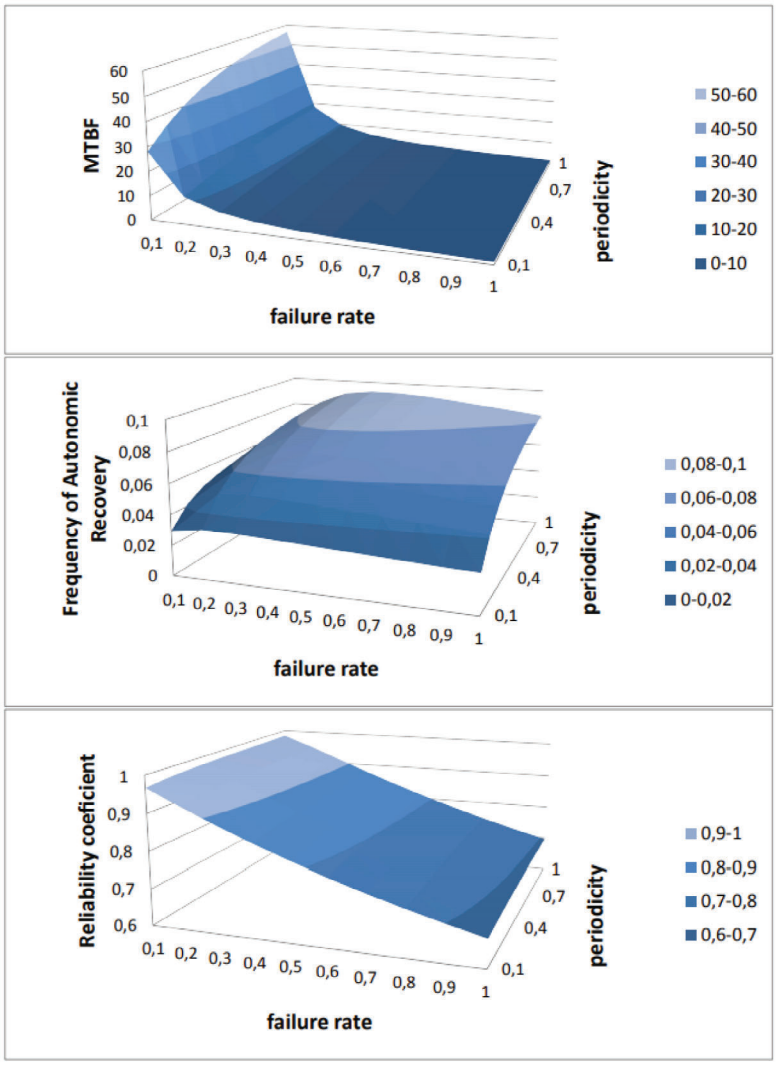

FIGURE 16 Reliability metrics

7.2.5 | Testing "System 3" example

Directed on system 3, previous experiments (i.e., different self-healing strategy) give same results evolution but to a lesser extent regarding the mean time response and the processed requests throughput.

Figure 18 shows a comparison between the results of the strategies of system 2 (strategy 1) and system 3 (strategy 2). Better results are obviously given by the first strategy that has two servers running continuously. However, accurate comparison according to each strategy's objective require more metrics.

7.2.6 | Testing "System 4" example

Figure 19 shows the mean response time of requests processed by the system, under the variation of monitoring thresholds. The threshold of monitored requests and the amount of scaled resources represent two main parameters affecting the autonomic manager efficiency. The figure shows an inversely logarithmic evolution of the response time, as the number of scaled resources increases, while the obtained results are slightly more important for a lower reconfiguration threshold. A stagnation is then noticed at the final part of the graph, where the response time remains slightly the same for the different threshold values. This metric helps in identifying configuration points where no impact is induced on system performances.

For the same parameters, Figure 20 gives the amount of resources used by each system configuration. This metric is a main value that should be optimized in a resource allocation system, as a high number of active resources increases the energy consumption and the system operation cost.

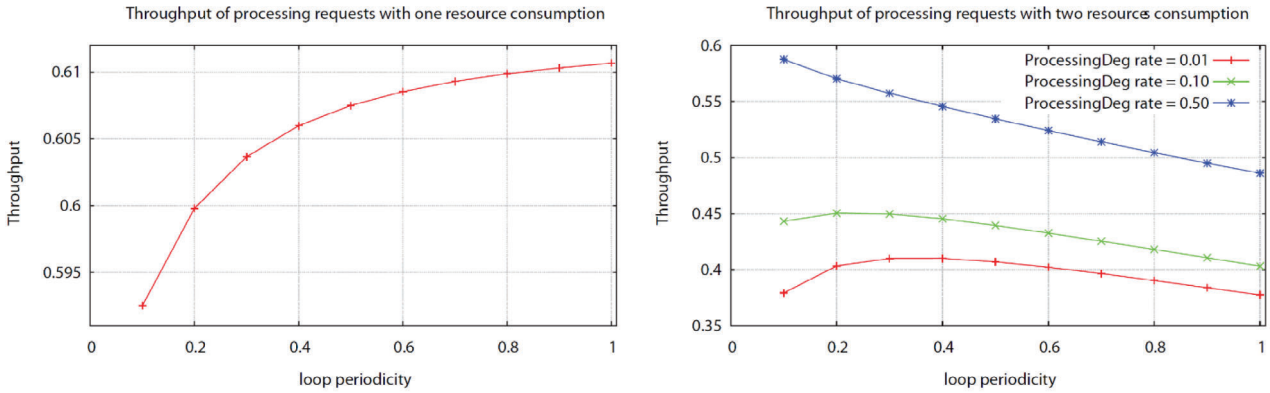

FIGURE 17 Throughput with resources consumption

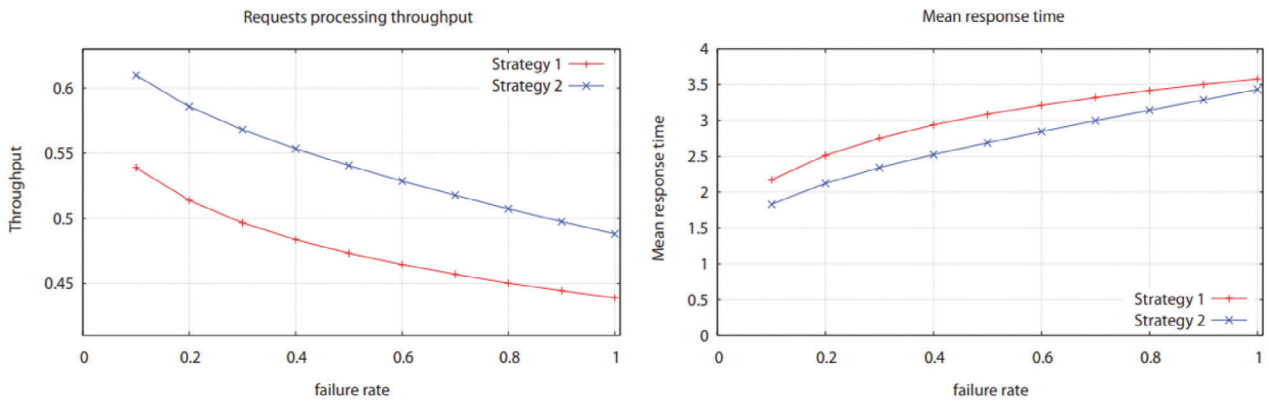

FIGURE 18 Comparison between the two strategies

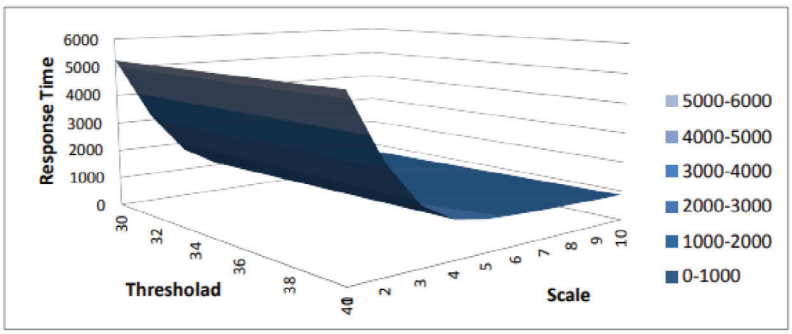

FIGURE 19 Mean response time when varying the autonomic manager parameters

First, a higher scaling implies obviously more resources use; Although, from a certain threshold of scaling, the consumption decreases and stagnate at a same point, this is due to performances improvement that leads to less reconfigurations, which implies less resources consumption. In the other hand, resource consumption decreases linearly according to the reconfiguration threshold increasing.

The last metric we interested in is the system's availability, as it is an important index that should be computed in order to define the system quality of service. This metric is also commonly considered in clients SLA and can lead to additional cost. Ensuring high availability is, so, a main issue when designing a system. Obtained results for our model, depicted in Figure 21, show a logarithmic evolution of the processed requests; when autonomic manager parameters give a slowly reactive system, a saturation state is quickly reached, an increasing resources scaling associated to a relatively small scaling threshold gives a maximal system availability.

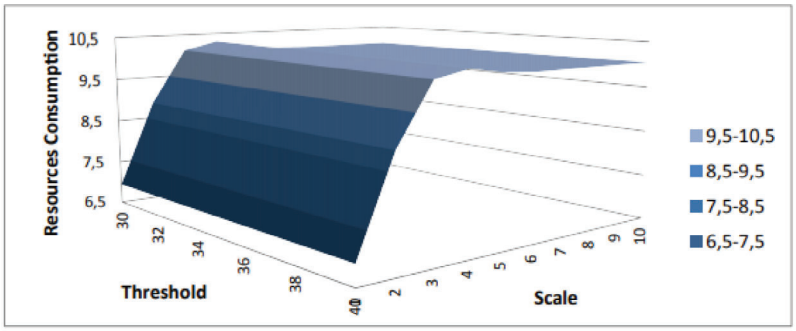

FIGURE 20 Resources consumption when varying the autonomic manager parameters

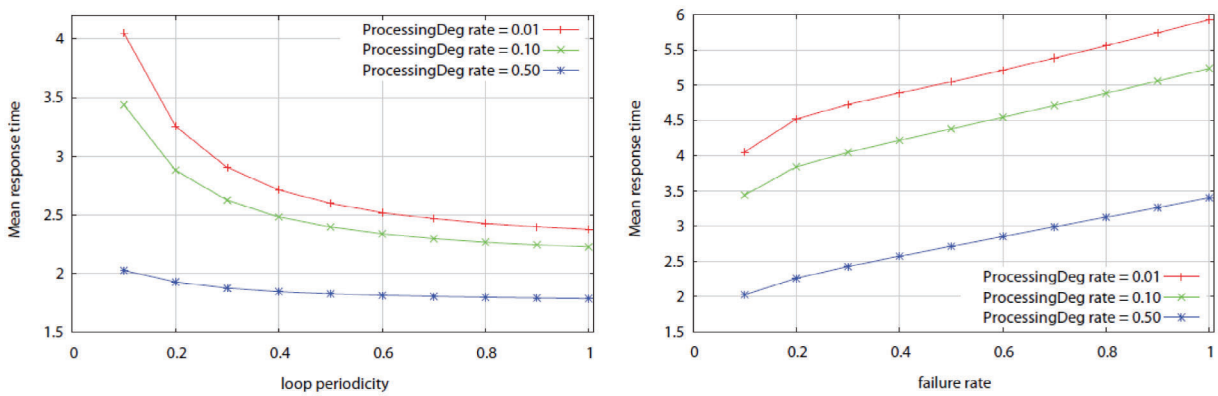

FIGURE 21 System's availability when varying the autonomic manager parameters

7.3 | Summary

Conducted experiments show that the use of our modeling approach helps in identifying suitable adjustments, according to workloads intensities and chosen parameters. The computed metrics can then be used to efficiently configure the targeted system, to ensure an appropriate solution resulting in a balance between costs, reliability and quality of service.

We conclude that estimating performances indexes depends on the model ability to represent the autonomic loop impact on the system, as shown with the introduction of resource consumption and different self-healing strategies. The regulation of the autonomic loop intervention has also to be considered, as this would reduce general performances.

Furthermore, to model this kind of distributed architectures, it is important to consider the following key points:

- The requests distribution representation
- The servers states
- Servers predispositions to survive failure and attacks

8 | CONCLUSION AND FUTURE WORKS

In this work, we addressed the issue of lack of a common and repeatable methodology allowing to forecast the performances of self-adaptive Cloud systems. Our contribution consists of a stochastic Petri nets modeling approach, oriented on resource allocation and self-healing in Cloud systems, and based on a new architecture description language.

We tried to include the most common parameters influencing the system and its environment change like workload variation, requests distribution and vulnerability to failure and cyberattacks. Moreover, we developed a tool automating the model generation based on the system description

with the suggested ADL. Our aim was to use the obtained models and their analysis, in order to forecast the best configuration to apply, and to compare different self-adaptation approach at design time.

During our experiments, the analysis of a complex SPN model showed some shortcomings as the performance computing of such model was resources intensive due to the steady state probability vector scale, which limits the level of detail that can be included in the system model. To overcome that scalability issue, we proposed the use of a Petri net reduction method to produce more flexible models and allow complex system modeling. Thanks to that, the experiments directed on our examples demonstrated the ability of our methodology to generate the performance forecasting results and highlight the differences between different self-adaptability strategies under different situations.

Another lack that needs to be addressed in our experiments as part of our future works is to apply our methodology on some existing resource allocation and self-healing approaches to compare the forecasted results with the authors results in order to validate the reliability of our methodology. We also need to refine the prerequisites to a self-adaptive approach to fit our methodology. An extension of our methodology allowing to convert a self-adaptability strategy toward a simplified model might be an interesting orientation to address.

Furthermore, we aim at applying our modeling approach on a Cloud infrastructure simulator, to enable us to evaluate our forecasted results with the ones generated from the simulator. Finally, we also plan to extend our methodology to container orchestration systems like Kubernetes or Docker Swarm that are more commonly used in AWS, Microsoft Azure and Google Cloud for providing containers autoscaling and self-healing across clusters.

DATA AVAILABILITY STATEMENT

Data sharing not applicable to this article as no datasets were generated or analysed during the current study

ORCID

Mehdi Sliem https://orcid.org/0000-0001-8256-1562

REFERENCES

1. Dehraj P, Sharma A. A review on architecture and models for autonomic software systems. *J Supercomput*. 2021;77(1):388-417.
2. Dehraj P, Sharma A. An empirical assessment of autonomicity for autonomic query optimizers using fuzzy-AHB technique. *Appl Soft Comput*. 2020;90:106137.
3. Litoiu M. A performance analysis method for autonomic computing systems. *ACM Trans Auton Adapt Syst*. 2007;2(1):3.
4. Manuel S, Exposito E, Aguilar J. Implementing self-* autonomic properties in self-coordinated manufacturing processes for the industry 4.0 context. *Comput Ind*. 2020;121:103247.
5. Filho RR, Porter B. Autonomous state-management support in distributed self-adaptive systems. Proceedings of the 2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C); 2020; 176-181.
6. Kinneer C, van Tonder R, Garlan D, Le Goues C. Building reusable repertoires for stochastic self-* planners. Proceedings of the 2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS); 2020:222-231.
7. Najme M, Ghafari R, Behnam MHZ. Mohammad Hasani Zade: "cloud computing simulators: a comprehensive review". *Simul Model Pract Theory*. 2020;104:102144.
8. Eisa M, Younas M, Basu K, Awan I. Modelling and simulation of QoS-aware service selection in cloud computing. *Simul Model Pract Theory*. 2020;103:102108.
9. Al-Shara Z, Alvares F, Bruneliere H, Lejeune J, Prud'homme C, Ledoux T. CoMe4ACloud: an end-to-end framework for autonomic cloud systems. *Future Gener Comput Syst*. 2018;86:339-354.
10. Abid R, Salaün G, De Palma N. Asynchronous synthesis techniques for coordinating autonomic managers in the cloud. *Sci Comput Program*. 2017;146:87-103.
11. Sliem M, Salmi N, Ioualalen M. An approach for performance modelling and analysis of multi-tiers autonomic systems. Proceedings of the 28th IEEE International Conference on Advanced Information Networking and Applications Workshops; 2014:352-357.
12. Mehrotra R, Srivastava S, Banicescu I, Abdelwahed S. An interaction balance-based approach for autonomic performance management in a cloud computing environment. *Adapt Res Manag Schedul Cloud Comput*. 2014;52-70.
13. Amoretti M, Zanichellib F, Conte G. Efficient autonomic cloud computing using online discrete event simulation. *J Parallel Distrib Comput*. 2013;73(6):767-776.
14. Maggio M, Hoffmann H, Papadopoulo A, et al. Comparison of decision-making strategies for self-optimization in autonomic computing systems. *ACM Trans Auton Adapt Syst*. 2012;7(4):1-32.
15. Shahidinejad A, Ghobaei-Arani M, Masdari M. Resource provisioning using workload clustering in cloud computing environment: a hybrid approach. *Clust Comput*. 2020;24(1):319-342.
16. Chen X, Wang H, Ma Y, Zheng X, Guo L. Self-adaptive resource allocation for cloud-based software services based on iterative QoS prediction model. *Future Gener Comput Syst*. 2020;105:287-296.
17. Nouri SMR, Li H, Venugopal S, Guo W, He MY, Tian W. Autonomic decentralized elasticity based on a reinforcement learning controller for cloud applications. *Future Gener Comput Syst*. 2019;94:765-780.
18. Mohit K, Sharma SC, Anubhav G, Singh SP. A comprehensive survey for scheduling techniques in cloud computing. *J Netw Comput Appl*. 2019;143:1-33.
19. Al-Asaly MS, Hassan MM, Alsanad A. A cognitive/intelligent resource provisioning for cloud computing services: opportunities and challenges. *Soft Comput*. 2019;23(19):9069-9081.
20. da Rosa Righi R, Lehmann M, Gomes MM, et al. A survey on global management view: toward combining system monitoring, resource management, and load prediction. *J Grid Comput*. 2019;17(3):473-502.

21. Gill SS, Buyya R. Resource provisioning based scheduling framework for execution of heterogeneous and clustered workloads in clouds: from fundamental to autonomic offering. *J Grid Comput*. 2019;17(3):85-417.
22. Ghobaei-Arani M, Jabbehdari S, Pourmina MA. An autonomic resource provisioning approach for service-based cloud applications: a hybrid approach. *Future Gener Comput Syst*. 2018;78:191-210.
23. Gill SS, Chana I, Singh M, Buyya R. CHOPPER: an intelligent QoS-aware autonomic resource management approach for cloud computing. *Clust Comput*. 2018;21(2):1203-1241.
24. Rahmanian AA, Ghobaei-Arani M, Tofighy S. A learning automata-based ensemble resource usage prediction algorithm for cloud computing environment. *Future Gener Comput Syst*. 2018;79:54-71.
25. Ataie E, Entezari-Maleki R, Etesami SE, Egger B, Ardagna D, Movaghar A. Power-aware performance analysis of self-adaptive resource management in IaaS clouds. *Future Gener Comput Syst*. 2018;86:134-144.
26. Nikravesh AY, Ajila SA, Lung C-H. An autonomic prediction suite for cloud resource provisioning. *J Cloud Comput*. 2017;6(1):3.
27. Al-Sharif ZA, Jararweh Y, Al-Dahoud A, Alawneh LM. ACCRS: autonomic based cloud computing resource scaling. *Clust Comput*. 2017;20(3):2479-2488.
28. Sliem M, Salmi N, Ioualalen M. Towards modelling-based self-adaptive resource allocation in multi-tiers cloud systems. *Int Conf Internet Distrib Comput Systems*. 2015;9258:11-18.
29. Yuan H, Bi J, Li BH, Chai X. An approach to optimized resource allocation for cloud simulation platform. *Commun Comput Inf Sci*. 2014;474:139-152.
30. Dong H, He B, Miao C. A survey of resource management in multi-tier web applications. *IEEE Commun Surv Tutor*. 2014;16(3):1574-1590.
31. Tchana A, Tran GS, Broto L, De Palma N, Hagimont D. Two levels autonomic resource management in virtualized IaaS. *Future Gener Comput Syst*. 2013;29(6):1319-1332.
32. Verhoefa C, Bhulaib S, Van der Mei R. Optimal resource allocation in synchronized multi-tier internet services. *Perform Eval*. 2011;68(11):1072-1084.
33. Chopra I, Singh M. SHAPE—an approach for self-healing and self-protection in complex distributed networks. *J Supercomput*. 2013;67(2):585-613.
34. Psaier H, Dustdar S. A survey on self-healing systems: approaches and systems. *Computing*. 2010;91(1):43-73.
35. Brown AB, Redlin C. Measuring the effectiveness of self-healing autonomic systems. Proceedings of the Second International Conference on Autonomic Computing (ICAC05); 2005:328–329.
36. Taherizadeh S, Grobelnik M. Key influencing factors of the Kubernetes auto-scaler for computing-intensive microservice-native cloud-based applications. *Adv Eng Softw*. 2020;140:102734.
37. Ragmani A, Elomri A, Abghour N, Moussaid K, Rida M, Badidi E. Adaptive fault-tolerant model for improving cloud computing performance using artificial neural network. *Proc Comput Sci*. 2020;170:929-934.
38. Hasan M, Goraya MS. Fault tolerance in cloud computing environment: A systematic survey. *Comput Ind*. 2018;99:156-172.
39. Ammendolaa R, Biagionib A, Frezzab O, et al. A hierarchical watchdog mechanism for systemic fault awareness on distributed systems. *Future Gener Comput Syst*. 2015;53:90-99.
40. Ricardo J Rodrguez CT, Merseguer J. Fault-tolerant techniques and security mechanisms for model-based performance prediction of critical systems. Proceedings of the 3rd International ACM SIGSOFT Symposium on Architecting Critical Systems - ISARCS '12; 2012:21-30.
41. Yang B, Tan F, Dai YS. Performance evaluation of cloud service considering fault recovery. *J Supercomput*. 2011;65(1):426-444.
42. Porter B, Filho RR, Dean P. A survey of methodology in self-adaptive systems research. Proceedings of the 2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS); 2020:168-177.
43. Montemaggio A, Iannucci S, Bhowmik T, Hamilton J. Designing a methodological framework for the empirical evaluation of self-protecting systems. Proceedings of the 2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C); 2020:218-223.
44. Zuefle M, Bauer A, Lesch V, et al. Autonomic forecasting method selection: examination and ways ahead. Proceedings of the 2019 IEEE International Conference on Autonomic Computing (ICAC); 2019:167-176.
45. Calinescu R, Mirandola R, Perez-Palacin D, Weyns D. Understanding uncertainty in self-adaptive systems. Proceedings of the 2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS); 2020:242-251.
46. Murata T. Petri nets: properties, analysis and applications. *Proc IEEE*. 1989;77(4):541-580.
47. Baarir S, Beccuti M, Cerotti D, De Pierro M, Donatelli S, Franceschinis G. The GreatSPN tool: recent enhancements. *ACM SIGMETRICS Perform Eval Rev*. 2009;36(4):4-9.
48. Firas D, Ahmed MAM. Toward agent-based petri net decision making modelling for cloud service composition: a literature survey. *J Netw Comput Appl*. 2018;130:14-38.
49. Thomas M, Chen JCS-A, Buford J. Petri net modeling of cyber-physical attacks on smart grid. *IEEE Trans Smart Grid*. 2011;2(4):741-749.
50. Choppy C, Klai K, Zidani H. Formal verification of UML state diagrams: a petri net-based approach. *ACM SIGSOFT Softw Eng Notes*. 2011;36(1):1-8.
51. Staines TS. Rule based Bi-directional transformation of UML2 activities into petri nets. *Int J Comput*. 2008;5(2):281-288.
52. Thong WJ, Ameedeen MA. A survey of petri net tools. *Lect Notes Electr Eng*. 2014;315:537-551.
53. Chiola G, Franceschinis GA, Gaeta R, Ribaudo M. GreatSPN 1.7: graphical editor and analyzer for timed and stochastic petri nets. *Perform Eval*. 1995;24(1–2):47-68.

How to cite this article: Sliem M, Salmi N, Ioualalen M. Designing a methodological framework for modeling and performance forecasting of self-adaptive cloud systems. *Concurrency Computat Pract Exper*. 2021;e6569. https://doi.org/10.1002/cpe.6569