Using Performance Modelling for Autonomic Resource Allocation Strategies Analysis

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

Nabila Salmi
MOVEP Laboratory, USTHB
Algiers, Algeria,
LISTIC, Université de Savoie
Annecy le Vieux, France
nsalmi@usthb.dz

Malika Ioualalen
MOVEP Laboratory, USTHB
Algiers, Algeria,
mioualalen@usthb.dz

***Abstract*—Distributed resource allocation in data centers has gained a lot of attention from the research community in the last few years, especially in fields like cloud computing and multitier systems. It is usually expected that these systems deliver some performance guarantees to users' Service Level Agreements (SLAs). Therefore, data center servers may need to be dynamically redeployed to optimize some performance metrics so that to meet the promised SLAs. Moreover, the total profit of a system depends on its ability to reduce a data center's energy cost through the resources utilization optimization. The main challenge of resource allocation is then to find the minimum amount of resources that an application needs to meet the desired Quality of Service (QoS). In this direction, autonomic computing appears to be one of the most popular concepts to achieve these goals by means of self optimization. These properties provide a system with a dynamic optimization of its own resources use, and enable it to autonomously adapt itself to its environmental changes. However, such autonomic resource allocation strategies may result in a loss of performance or even service degradation under some conditions. Furthermore, it is interesting to predict the behaviour and the efficiency of those strategies, before applying a new resource allocation, to forecast the most appropriate configuration and ensure the effectiveness of the autonomic manager. Thus, we propose in this paper a general insight of performance modelling of resource allocation strategies using the modelling of an autonomic resource allocation server as an example. The modelling is based on stochastic Petri net models (SPN). We consider in our modelling dynamic allocation strategies, based on workload intensity and user mixes. Finally, we illustrate the effectiveness of our modelling through a set of experimental results.**

Keywords–Autonomic computing; data center; performance modelling; resource allocation

I. INTRODUCTION

Todays data centers are becoming increasingly large and complex, hosting a variety of business-critical applications with a set of QoS requirements, such as those for web hosting or e-commerce sites. The increasing demand for computing resources in a shared infrastructure creates the challenge of dynamic on-demand resource provisioning and allocation in response to variable workloads [1].

Data centers need then to have more flexible execution environments, allowing resources sharing between its different applications in order to meet performances requirements of those applications. In a cloud computing application for instance, the main objective is to maximize profits by an efficient use of resources, such as meeting the clients SLAs and reducing the energy cost of the data centre, by an efficient use of resources. Furthermore, modern Internet applications are implemented on multi-tier architectures, increasing the application's complexity. Each tier provides a defined service to the next tier and uses services from the previous tier. The resource allocation problem for multi-tiers applications is then harder than that for single tier applications: tiers may not be homogeneous and a performance bottleneck in one tier may decrease the overall profit.

The key-challenge of resources allocation is then, to provide enough resources to an application to meet its performance goals while avoiding an over-provisioning that could increase the energy cost and reduce the efficiency for other concurrent hosting (i.e., less resources for next applications). Due to these economic benefits, the resource provisioning optimization has been the subject of much investigations [2].

Some approaches focused on static allocation strategies that consider a fixed set of applications and resources, but these approaches have shown their weak efficiency because of the changing workload mixes. Other approaches are based on peak demand but suffer too from a lack of efficiency and a non cost-effectiveness due to their poor resource utilization during off-peak periods. In contrast, autonomic resource management may lead to efficient resource utilization and fast response in the presence of changing workloads.

In cloud applications, autonomic resource allocation provides application environments with self-configuration and self-optimization capabilities according to their environmental changes. The system can then be enforced through scale-up/down (i.e., adding/removing resources to individual Virtual Machines (VM)), scale-out/in (i.e., adding/removing VMs to an application environment), and migration (i.e., moving VMs over the physical infrastructure). This will directly impact both applications performances and the providers operation cost.

To achieve those autonomic computing features, IBM suggested a reference model for autonomic control loops, which is sometimes called the MAPE-K (Monitor, Analyse, Plan, Execute, Knowledge) loop [3]. The architecture dissects the loop into four parts that share knowledge:

- The *monitor* function provides mechanisms that collect, aggregate, filter and report details (such as metrics and topologies) collected from a managed resource.
- The *analyze* function correlates and models complex situations (for example, time-series forecasting and queuing models). The provided mechanisms allow the autonomic manager to learn about the Information Technology (IT) environment and help in predicting future situations.

 ISBN: 978-1-61208-331-5

- The *plan* function provides mechanisms that construct actions needed to achieve goals and objectives. The plan mechanism uses policy information to guide its work.
- The *execute* function provides launches the execution of a plan and controls it with considerations for dynamic updates.

Even if self-optimization may appear as an attractive way to enforce resource allocation process, reaching the goal of system improvement varies from an autonomic system to another, depending on the autonomic manager architecture and its implemented features. The targeted function to improve and the resource allocation process are also determinant criteria in building an efficient autonomic system. Furthermore, autonomic features are costly: they make systems more and more complex and need considerably more resources than other systems. So, attempting to improve a system with autonomic features may lead to some undesired configuration, with the introduction of new bugs or the loss of some vital settings, or else the degradation of the resulted configuration performances.

Hence, it is important to ensure that the chosen allocation strategy offer the desired performances under different circumstances while designing the system. This requires to predict and measure performances of an autonomic manager and its possible impact on the current system, before applying a solution or a reconfiguration. The main goal is to know how long the decision making process will take and how much the system's performances will be improved. The autonomic resource allocation has also to be compared with static resource allocation according to the application case. In this direction, formal methods are strong tools for system performance prediction based on modelling. Mathematical models, such as Petri nets [4] are well suitable for modelling distributed systems and fit the autonomic computing process with the system operation.

We attempt, in this paper, to define a general modelling of autonomic resource allocation using Stochastic Petri Nets (SPN). This modelling is presented through a simple typical example of an autonomic resource allocation system. We consider for our modelling a dynamic allocation strategy, based on workload mixes analysis, applied by an autonomic server. Some static allocation strategies are defined, each one assigning a fixed amount of resources to user requests. The system will work initially with a predefined strategy, while the autonomic manager analyzes continuously workload mixes. The autonomic manager will, then, reconfigure the system to move to a more appropriate strategy, whenever there are pending user requests while other requests hold several resources. So, according to the analysis of the monitored data, the autonomic manager redistributes fairly the existent resources.

The paper is organized as follows: Section II discusses related work. Then, Section III presents our general modelling and the experimental results. Finally, Section IV concludes the paper and gives future work.

II. Related Work

Significant attention has been given to the topic of distributed resource allocation in the last few years. The main studied issue is the cost, efficiency and the generated profits of the used methods, especially when client Service Level Agreements (SLAs) must be satisfied. Several work have then been proposed to use autonomic computing, while using predictive models. We provide, in the following, the most relevant prior work.

As Clouds are complex, large-scale, and heterogeneous distributed systems, resource allocation is one of the main topics of interest studied in the last few years in the context of cloud computing. Thus, Ghanbari et al.'s, results [2] provide valuable insights on the performance of alternative resource allocation strategies and job scheduling disciplines for a cloud computing infrastructure. The service level agreement is based on a response time distribution, which is more relevant than the mean response time with respect to performance requirements of interactive applications. The authors developed an efficient and effective algorithm to determine the allocation strategy that results in a smallest number of required servers. They have also developed a novel scheduling discipline, called probability dependent priority, which is superior to First Come First Served (FCFS) and head-of-the-line priority in terms of requiring the smallest number of servers. The authors consider in their work the case of two job classes.

In the same direction, Buyya et al. [5] identifies open issues in autonomic resource provisioning and presents innovative management techniques for supporting Software as a Service (SaaS) applications hosted on Clouds. The authors present a conceptual architecture and early results highlighting the benefits of Clouds autonomic management. They presented the first steps towards an autonomic Cloud platform able to handle many of the above problems. Such a platform will be able to dynamically supply applications with Cloud resources in such a way that Quality of Service user expectations are met with an amount of resources that optimizes the energy consumption required to run the application.

In [6], an SLA-based resource allocation problem for multi-tier applications in the field of cloud computing is considered by Goudarzi and Pedram. An upper bound is provided on the total profit and an algorithm based on force-directed search is proposed to solve the problem. Processing, memory requirements and communication resources are considered as three dimensions in which optimization is performed. In [1], the purpose was to demonstrate the advantage of "adaptive" models, relative to "static" models in optimization. Hu et al. investigated model based optimization of a private cloud where applications are clustered across a known homogeneous set of physical machines. They modified resource sharing of applications, to minimize SLA violations. The focus was only on response time, considering that multiple service level objectives will not change the approach, but just the complexity of solving the optimization problem. The main contribution of this work was using dynamically tracking models (for each application) within the global optimization loop. These models update themselves at runtime in order to adapt to environment perturbations, not captured in initial model specification.

Workload variation and its resource consumption is also an important point to study for the resource management, in this direction, Litoiu [7] investigates performance analysis techniques to be used by the autonomic manager. The workload complexity was studied, and algorithms were proposed for computing performance metrics bounds for distributed transactional systems under asymptotic and non-asymptotic

 ISBN: 978-1-61208-331-5

conditions, with saturated and non-saturated resources respectively. The proposed technique makes use of linear and non-linear programming models and their performance evaluation. Workloads are characterized by their intensity representing the total number of users in the system, and workload mixes, which depict the number of users in each service class.

Finally, some authors investigate the use of mathematical and performance modelling or optimization approaches to improve the resource allocation. Bennani and Menasce [8] addressed the resource allocation problem in autonomic data centers. The presented solution is based on the use of analytic queuing network models combined with combinatorial search techniques. The authors have shown how analytic performance models can be used in an efficient manner, to design controllers that dynamically switch servers from one application environment to another as needed.

In [9], Xu et al. propose a two-level resource management system to dynamically allocate resources to individual virtual containers. It uses local controllers at the virtual-container level and a global controller at the resource-pool level. An important advantage of this two-level control architecture is that it allows independent controller designs for separately optimizing applications performances and the resources use. Autonomic resource allocation is realized through the interaction of the local and global controllers. A novelty of the local controller designs is their use of fuzzy logic-based approaches to efficiently and robustly deal with the complexity and uncertainties of dynamically changing workloads and resource usage. The global controller determines the resource allocation based on a proposed profit model, with the goal of maximizing the total profit of the data center.

Regarding these proposals, we notice that most of them use formal modelling for an autonomic online optimization, but only few work focused on the efficiency of those autonomic components and performance modelling and prediction of a whole autonomic system in the context of resource allocation.

We introduce, then, in this paper, a general performance modelling and analysis approach for the autonomic resource allocation problem. The main idea is to model the complete autonomic system behaviour including resource management and allocation and the autonomic loop. A system configuration is seen as a distribution of system resources for user requests and their availability. We illustrate then our modelling methodology through a simple typical example of an autonomic server and a series of experimental results.

III. An autonomic resource allocation Platform

To explain our modelling approach, we choose a typical example of an autonomic resource allocation system, based on a simple monitoring process. We first, in this section, define the architecture of our example, then we give its general functionalities which have to be considered in our modelling. After that,we explain our modelling methodology through the presentation of our model based on Stochastic Petri Nets (SPN) [10].

A. System architecture

Our system consists of an autonomic server with a set of resources and an allocation strategy for those resources based on a monitoring process. The monitoring process aims to determine the amount of resources to allocate to the next user requests.

The main managed element in an autonomic resource allocation system is the resource. A resource may be any element used or invoked during the processing of a user request: it may be a server, a processor cycle, a memory space, a used device, a network bandwidth, an available component, and so on. An allocation strategy defines the amount of resources to allocate to a new requests: it could be a static allocation approach considering a fixed set of resources or a dynamic one adapting the resource management strategy according to the system's state and the user SLAs.

The autonomic loop aim at self-optimize system performances through self-configurations, by switching between allocation strategies relatively to the changing workload. We take for our example a simple self-optimization technique based on the current workload mixes, a continuous monitoring of processing services is done by the autonomic manager, the autonomic loop is, then, triggered periodically to analyze the monitored data, the analysis and plan phases determine the most present service's class in the system and choose the most appropriate strategy for this current class for the next requests. Finally, the act phase switches from the current strategy to the new one and reset the autonomic loop periodicity.

B. System features to model

To show how we operate to model an autonomic system, we take generic concepts of an autonomic resource allocation system. However, to have a reliable trustworthy model with accurate results, some key concepts have to appear in our modelling :

a) System's resources and allocation strategies: We need, in our modelling, to represent the system resources, their states and their distribution over the time and their allocation mode. A resource may be a server, a processor cycle, a memory space, a network bandwidth, and so on. A system configuration is then seen as a state where resources are allocated to different user requests according to a predefined strategy.

For instance, the system may reach or approach a saturation state, becoming unefficient to process new requests; it may also be in a an unoptimized state not satisfying a required SLA, even if the available resources are sufficient. It may also give the expected processing performance while producing a high consumption cost. All these drawbacks or failures depend on the allocation strategy and its efficiency to optimally distribute the system's resources.

b) Classes of service: As presented in [7], the system performance and the saturation state do not depend only on the workload intensity (number of users in the system), but also on the workload mixes that represent the users number of each service class. Classifying user requests into different classes allows us to separate them according to a set of specificities that may affect differently the system behaviour and its performances. In fact, a request which needs a database access in a multi-tier system do not have the same impact on the system's performance as a request invoking only an application process in the same system. In our work, we consider different service classes classified according to the needed number of identical resources (i.e., several CPUs or

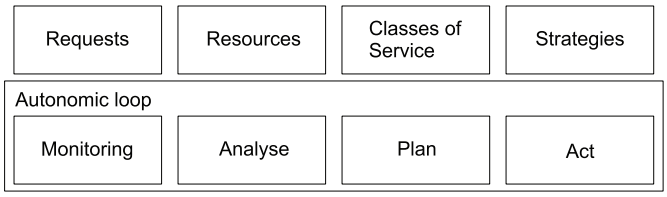

Figure 1: Key concepts to appear in the model

more memory space): each class requires a given number of resources. This consideration comes from the fact that a job holding a set of resources ends more quickly than if it had only one resource. Contrariwise, allocating more than needed resources to a client will waste additional resources for the same job performances while increasing costs and reducing concurrency processing for other jobs. Hence, resource provisioning and the resulting system configuration vary according to the workload mixes in the system.

c) Self-optimization strategy: As explained before, an allocation strategy have to be defined according to the workload mixes. Modelling static and autonomic self-optimization approaches will allow to compare the efficiency and inconveniences of each resource management manner for each system configuration. Hence, for an autonomic approach, it is important to represent the system and its autonomic loop in the same model, this allows to measure autonomic features impact on the system while measuring the system performance itself. Grouping the system functioning and the autonomic loop in the same model is also more appropriate to make tests under different cases: different system configurations can then be used to test the autonomic loop influence; the system's state and its evolution over time being visible directly in the model.

C. Modelling our autonomic architecture

1) Used formalism: Stochastic Petri Nets: Stochastic Petri Nets (SPN) [4] are a stochastic temporal extension of Petri Nets, widely used for performance analysis of complex systems. Our choice of this formalism is first motivated by the fact that we need a state based model to be able to evaluate performance indices related to system configurations (number of requests in some part of the system, mean usage time of some resource, etc.). Petri Nets are state based models, which are well known for being able to model complex systems with concurrency and conflicts, and the stochastic extension allows to do a performance analysis based on a Markov chain-like state space graph. This is in contrast to other performance formalisms like Queuing networks or process algebras models where conflicts cannot be modelled. Moreover, as our methodology is incremental, we need to compose sub-models to connect multi-tier sub-models, or servers sub-models. In this direction, Petri Nets are compositional models and interaction between Petri nets representing different parts of a system may be easily defined as transitions or places, which are merged when interacting sub-models are composed. So, we define an SPN in the following.

Definition 1 (Stochastic Petri Net)**.** *A Stochastic Petri Net is a couple $\mathcal{N} = (\mathcal{N}'; \theta)$ where:*

- *$\mathcal{N}' = (P, T, Pre, Post, Inh, Pri)$ 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 the 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, i.e, the parameter of its exponential law, where $\theta(t)$ represents:*
  - *The weight of t if $Pri(t) > 0$ (t is immediate).*
  - *The firing rate of t if $Pri(t) = 0$ (t is timed): the enabling duration before the firing of $t(c, M)$ follows an exponential probability distribution with mean $\theta(t)$.*

2) System's modelling: Regarding to the system features given before, we follow the principles below to model the autonomic resource allocation system:

- We first model the requests arrival, a request being any kind of a client service invocation, applied to different data center applications.
- We then represent system resources, where we abstract the kind of resources as for requests.
- Resources are allocated to requests according to an allocation strategy. As in our system, different strategies are considered, this requires to model each strategy. However, only one strategy have to be applied at one time.
- After the allocation of the required resources to a request, the service processing in progress is modelled, as well as the different service classes to represent resources consumption and the obtained performances
- Finally, as our example is based on an autonomic system, we need to model each phase of the autonomic loop: the continuous monitoring of processing services, the autonomic loop triggering, the analysis and plan phases, which chooses the most appropriate strategy for the next requests, and the act phase which switches from the current strategy to the new one.

The modelling methodology explained above is depicted in Figure 1, giving the skeleton of a general resource allocation autonomic system model. This figure shows the key concepts that should be considered when modelling the system. Each part of the system is then replaced with the appropriate sub-model according to the real system to analyze. The merging of the obtained sub-models gives the whole model of the autonomic system. We give next the proposed sub-models for an autonomic server.

Figure 2: The request submitting sub-model

Figure 3: The resource allocation strategies sub-model

a) The request submitting sub-model: The request is modelled with a place named $Requests$, marked with a number of tokens corresponding to a possible number of user requests. The system processes a request (represented by a token) and answers the user, then the token is put back into the $Requests$ place to model a new request arrival and keep the testing possibility of the whole system. The transition $NewReq$ is used to model the arrival and queueing of the request in the system (see figure 2). This transition is characterized with a firing rate modelling the arrival request rate.

b) The resource allocation strategies sub-model: Each allocation strategy is modelled by a place. We consider in our model, three strategies: the first one for allocating one resource to new requests, modelled by a place named $Strat-1$; the second one for allocating two resources modelled by a place named $Strat-2$; and the last one for allocating three resources, represented by a place named $Strat-3$. This model is shown in Figure 3.

c) Servers sub-model: The model of servers is composed of two parts:

- The server requests queue part: It is modelled by a first place, named $Queue$, connected to the $Requests$ place. The requests are sorted into different classes to represent different execution processing times, giving a number of places equal to the number of service classes: we associate a place for each request class to model resource allocation to requests of the corresponding class.
- The servers parts: each server of the system is modelled by a place, named $Server$, containing the server's resources (see Figure 4). The request processing is modelled by three transitions and three places for each service class: The transitions are named $Begin-Cli-1$, $Begin-Cli-2$, $Begin-Cli-3$, representing the processing beginning for each current allocation strategy; The places, named $Cl1i-1-exe$, $Cli-2-exe$, $Cli-3-exe$ model the different request execution states. Three other transitions, named $End-Cli-1$, $End-Cli-2$, $End-Cli-3$, are used to model the request processing end (see Figure 4). The $Begin-Cli-s$ transitions are linked to the $Server$, $Queue$, $C1i-exe$ and $Strat-i$ places. The $End-Cli-s$ transition consumes resource tokens from the $Cli-s-exe$ place, and puts them back in the $Server$ place, and by the way, puts back the request to the $Requests$ place. This server sub-model represents a request processing according to a predefined execution time, given by the firing rates associated to the $Begin-Cli-s$ transition. These rates are fixed before analyzing the model, according to the system under test. As many user classes are defined in our model, this sub-model is duplicated for each user class with a slight modification of the required resources number to consume to get the best performances. Different processing rates are associated to their corresponding transitions, to show a better processing time when using more resources. Hence, only one beginning transition can be fired at one time in the same service class, according to the current strategy.

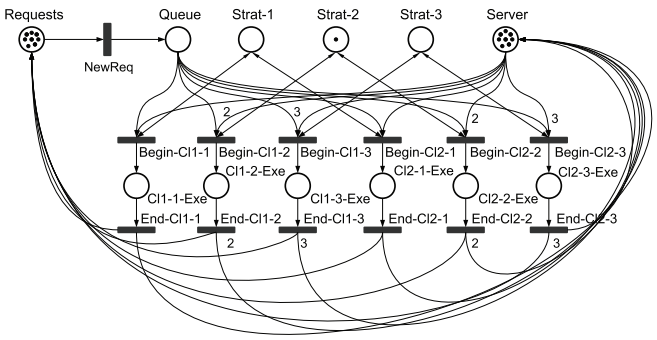

Figure 4: The server sub-model

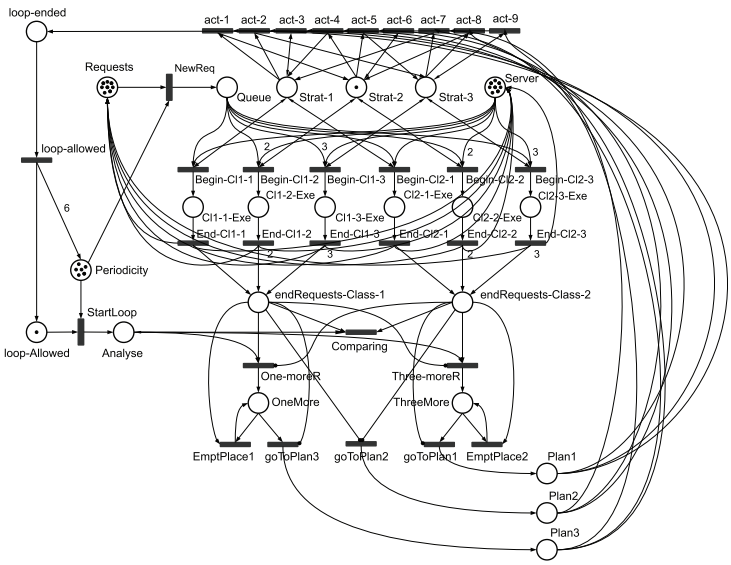

Figure 5: Global system model

d) The autonomic loop sub-model: The MAPE-K loop in our modelling is composed of five parts:

- The first part models the monitoring phase of the autonomic loop. As in our example, the self-optimization is based on the amount of requests in each service class, we model the monitoring by a place named, $endRequests-Class-i$, for each service class, each of these places is connected to $End-Cli-s$ places belonging to the corresponding service class (see Figure 5).
- Unlike the monitoring phase, which is performed continuously, other phases of the autonomic loop are, in our example, triggered periodically. In Figure 5, a place, named $periodicity$, models the periodicity of the autonomic loop by its number of tokens. A token

is taken periodically by the $newRequest$ transition. The place named $loopallowed$ contains the token that will represent the progression of the MAPE-K loop execution through the model and ensure that is executed once at a time.

- The analyze phase sub-model aims to empty all monitoring places, to reset them for the next period while computing which service class is the most frequent in the system in the last period. Hence, a place, named $analyse$, models the beginning of the phase. A transition, named $comparing$, takes a token from each monitoring place until some of them become empty, the $one - moreR$ and $three - moreR$ transitions are then used to continue the same process for the remaining monitoring places containing tokens. Moreover, inhibitor arcs are used to guide the analyze phase towards the appropriate transitions, when some monitoring places are empty. Once only one place is containing tokens, $OneMore$, $EmptPlace1$ or $ThreeMore$, $emptPlace2$ places are used to reset the corresponding place to an empty state. Finally, transitions $goToPlan1$ and $goToPlan2$ guide the autonomic loop to the associated reconfiguration plan. The transition $goToPlan3$ is fired when both service classes get the same number of requests in the last period (see figure 5).

- The plan phase sub-model is consisting of a set of places whose number is equal to the number of service classes. Each place represents the most appearing class in the last period ($Plan1$, $Plan3$). Moreover, other places have to be added to the plan, to consider all equality cases between the class requests: in our example, the $Plan2$ place models the equality case between the two service classes of our system (see Figure 5).

- Finally, the last part of the MAPE-K loop to model is the act phase. We model it by a set of transitions representing all combinations between the current applied strategy and the current computed plan. The new strategy, indeed, have to be chosen based on these two parameters, for instance, if the current strategy allocates three tokens (resources) to user requests while these requests belong to the first service class, the current strategy is then considered as non-cost-efficient, as it allows two more resources for the same service performance. A unique transition may be then fired, switching the strategy to the more appropriate place $strat-1$. The same process is repeated for each combination by a particular transition $act-i$. A place $loop-ended$ is reached after the firing of one of the act transitions; the transition $loop-allowed$ reset, then the loop periodicity and puts back the loop progression token to it initial place ($loopallowed$).

Figure 5 shows the general model of the autonomic server. The obtained model can be tested using different number of classes and periodicities for each test, to get more accurate results or to identify the best requests classification. In the next section, we analyze the obtained SPN and try to predict the modelled server behaviour under different configurations.

TABLE I: TRANSITION RATES OF CLASSES OF SERVICE

Config	Transition	Rate value
1	Begin-Cl1-strat	1.00
1	Begin-Cl2-1	0.33
1	Begin-Cl2-2	0.66
1	Begin-Cl2-3	1.00

D. Experimental results

The final model presented in Section III was analyzed using the GreatSPN package [10], on an Ubuntu linux 12.4 LTS workstation with 4 GB of RAM.

For performance analysis, we used various configurations of the system, obtained by varying the initial markings of the requests number, the available resources and the loop periodicity when using dynamic strategy. The model used in our experiments contains two service classes: the first one needs one resource from the server's place, while the second one uses three resources. We performed our tests under each of the three static allocation strategies and a dynamic strategy using the autonomic loop.

Table I shows the transitions rates, depending on the number of consumed resources. Only the second service class transitions are affected by the available resources. Transitions not appearing in this table have rate 1 (i.e., faster than all other transitions, rates being given in the same unit).

To compute performances, we vary the number of requests and available resources under each configuration. The GreatSPN tool [10] computes the state space and its steady-state probabilities. We study the evolution of several metrics from obtained steady-state probabilities. To evaluate the efficiency of a configuration, we interested in the following metrics:

- The response time for a user request, being of the first or second class.
- The throughput of processed requests.
- The relationship between response time and resource consumption.

We first analyze the SPN for a fixed number of requests (10 in our example), with varying the resources number of the server (1 to 40). Figure 6 shows that the mean response time of the static strategies is unchanged under increasing the number of available resources, while dynamic resource allocation slightly improves it taking advantage from the powerful autonomic system. However, the response time of the autonomic strategy remains worse than static ones that allocate more resources.

Figure 6 depicts the throughput of processed requests. Static strategy allocating one resource gives the best throughput, but is exceeded by other static strategies allocating more resources from a certain threshold of available resources. The autonomic resource allocation gives the worse throughput, which is partly due to the autonomic loop processing searching the best strategy. The results can be improved using a separated server for the autonomic manager. The efficiency of a given strategy in our case depends, though, more on its ability to reduce the final system's cost.

We were also interested in evaluating the total resource consumption of the system. The results shown in Figure 6 show

 ISBN: 978-1-61208-331-5

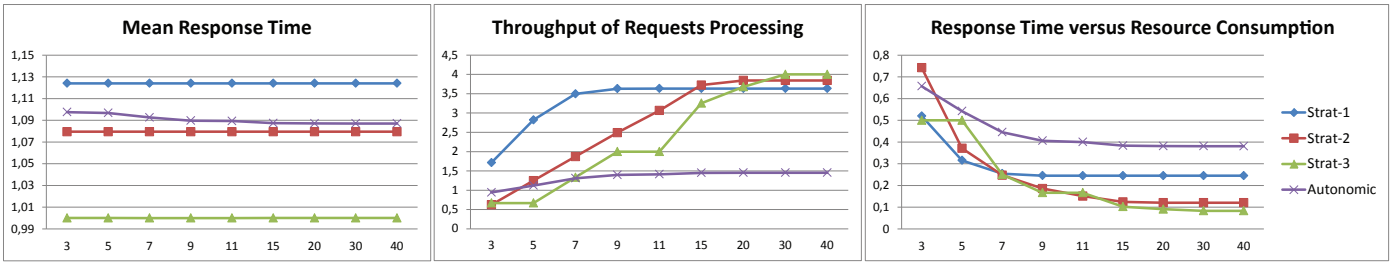

Figure 6: Experimental results depending on available resources

that the dynamic resource allocation gets the best ratio between the response time and the resource consumption metric, but it is also due to a less number of processed requests, as it is shown in Figure 6.

We can conclude that, with these different analysis results, an autonomic allocation strategy gives a better relationship between the response time and the resource consumption, and so less final user cost. It allows, though, to process less requests, which may make the system less effective. The most efficient strategy to use for a particular system, depends then, on the different costs related to each metric and the required expected quality of service.

IV. Conclusion and Future Work

This paper addresses a general modelling of resource allocation strategies for an autonomic server using SPN. The objective is to show, through the chosen example, how we can gradually build a formal model for a resource allocation autonomic system, to be able to analyze it and think about its performances and efficiency. For this purpose, we have studied the most important concepts to consider when modelling a particular system to obtain a reliable model, then construct the global model of the system example, basing on the building of sub-models representing parts of the system, then the merging of constructed sub-models.

To finalize the study, we compared different models of static and autonomic strategies with the aim of forecasting the more appropriate allocation strategy for the given system.

Regarding the obtained results of our modelling methodology, more research work is still required in several directions, among which: considering workload mixes in the request arrival modelling, modelling more specific applications of resource allocation systems, with specific requirements, such as a cloud computing system. Finally, more specific autonomic resource allocation strategies using different techniques have to be compared.

References

[1] Y. Hu, J. Wong, G. Iszlai, and M. Litoiu, "Resource provisioning for cloud computing," *Conference of the Center for Advanced Studies on Collaborative Research*, pp. 101–111, 2009.

[2] H. Ghanbari, B. Simmons, M. Litoiu, and G. Iszlai, "Feedback-based optimization of a private cloud," *IBM Canada*, vol. 28, pp. 10–111, January 2012.

[3] IBM, "An architectural blueprint for autonomic computing (fourth edition)," 2006.

[4] S. Natkin, "Stochastic petri nets and their application for computer systems evaluation," Ph.D. dissertation, CNAM, Paris, France, juin 1980.

[5] R. Buyya, R. N.Calheiros, and X. Li, "Autonomic cloud computing: Open challenges and architectural elements," *Third International Conference on Emerging Applications of Information Technology (EAIT)*, pp. 3–10, 2012.

[6] H. Goudarzi and M. Pedram, "Multi-dimensional sla-based resource allocation for multi-tier cloud computing systems," *IEEE 4th International Conference on Cloud Computing*, July 2011.

[7] M. Litoiu, "A performance analysis method for autonomic computing systems," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 2, 2007.

[8] M. N. Bennani and D. A. Menasce, "Resource allocation for autonomic data centers using analytic performance models," *2005 IEEE Intl'. Conf. on Autonomic Computing, Seattle, Washington*, pp. 229–240, June 2005.

[9] J. Xu, M. Zhao, J. Fortes, R. Carpenter, and M. Yousif, "Autonomic resource management in virtualized data centers using fuzzy logic-based approaches," *Cluster Comput'08*, vol. 11, pp. 213–227, 2008.

[10] S. Baarir, M. Beccuti, D. Cerotti, M. D. Pierro, S. Donatelli, and G. Franceschinis, "The greatspn tool: Recent enhancements," *ACM SIGMETRICS Performance Evaluation Review*, vol. 36, pp. 4–9, March 2009.

 ISBN: 978-1-61208-331-5