MODELING AND ANALYZING CLOUD AUTO-SCALING MECHANISM USING STOCHASTIC WELL-FORMED COLOURED NETS

Mohamed M. Ould Deye
Mamadou Thiongane
Mbaye Sene
Department of mathematics and computer science
Cheikh Anta Diop University
Dakar, Senegal
{mohamed.oulddeye, mamadou.thiongane, mbaye.sene}@ucad.edu.sn

KEYWORDS
Auto-scaling, Cloud computing, Stochastic Well-formed coloured Nets

ABSTRACT

Auto-scaling is one of the most important features in Cloud computing. This feature promises cloud computing customers the ability to best adapt the capacity of their systems to the load they are facing while maintaining the Quality of Service (QoS). This adaptation will be done automatically by increasing or decreasing the amount of resources being leveraged against the workload's resource demands. There are two types and several techniques of auto-scaling proposed in the literature. However, regardless the type or technique of auto-scaling used, over-provisioning or under-provisioning problem is often observed. In this paper, we model the auto-scaling mechanism with the Stochastic Well-formed coloured Nets (SWN). The simulation of the SWN model allows us to find the state of the system (the number of requests to be dispatched, the idle times of the started resources) from which the auto-scaling mechanism must be operated in order to minimize the amount of used resources without violating the service-level agreements (SLA).

INTRODUCTION

Cloud computing environments offer service such as processing, bandwidth, and storage. Customers rent these services to deploy their applications and guarantee a certain quality of service for end users. There are many important features of clouds computing but one of those features that has made these systems successful is obviously auto-scaling or elasticity. Auto-scaling is the process that automatically readjusts the cloud computing resources according to the current system load. This adaptation results in increase resources (scale out) when the workload grows, and in decrease resources (scale in) when the workload drops. The primary benefit of auto-scaling, when configured and managed properly, is that the workload gets exactly the cloud computational resources it requires (and no more or less) at any given time. Customers pay only for resources they need, when they need them. There are several techniques of auto-scaling for determining the appropriate moment to scale resource. We can cite for example the Application Profiling Technique (APT) (Hector et al. 2014; Qu et al. 2016; Sharma et al. 2011), the Static Threshold-Based Rules (STR) (Fallah et al. 2015; Han et al. 2012), the Time Series Analysis (TSA) (Kumar and Singh 2018; Roy et al. 2011), the Machine Learning (ML) (Tesauro et al. 2006), and the Queuing Theory (QT) (Villela et al. 2007).

APT is a process of finding maximum point of resource used by an application with a certain workload. It is a simple ways to find the desired resources at a different point of time. STR is the most popular technique. It defines threshold resources utilization to scale in or scale out. A simple example: if CPU > 80%, then scale out; if CPU < 20%, then scale in. It is quite difficult to set the correct thresholds and this must be done manually. In this paper we propose a method to find them. TS includes a number of methods that use a past history window of a given performance metric in order to predict its future values. Three methods are often considered: moving average, exponential smoothing and linear regression. This technique forecast the future workload and resources required. QT is one of the widely used for modeling Internet applications. It is used in the analysis phase of the auto-scaling process. It estimates the performance metrics and waiting time for the requests. Queuing theory is a field of applied probability to solve the queuing problem. ML is a technique that is used on online learning for the constructing dynamic approach for the estimation of resources. It is a self-adaptive technique as per the workload pattern available. See (Al-Dhuraibi et al. 2018; Singh et al. 2019) for more details in auto-scaling techniques.

There are also two types of auto-scaling: horizontal auto-scaling and vertical auto-scaling. Horizontal auto-scaling refers to adding more virtual machines to the auto-scaling group. Vertical auto-scaling means scaling by adding more power rather than more units, for example in the form of additional Core or CPU or RAM. Indeed, whatever the type and technique of auto-scaling used, it is difficult to determine at which state of the system (load, queue size, CPU % used, etc) the resources should be increased or decreased in order to

Communications of the ECMS, Volume 35, Issue 1,
Proceedings, ©ECMS Khalid Al-Begain, Mauro Iacono,
Lelio Campanile, Andrzej Bargiela (Editors)
ISBN: 978-3-937436-72-2 / 78-3-937436-73-9(CD) ISSN 2522-2414

minimize the resources used and guarantee the service-level agreements (SLA).

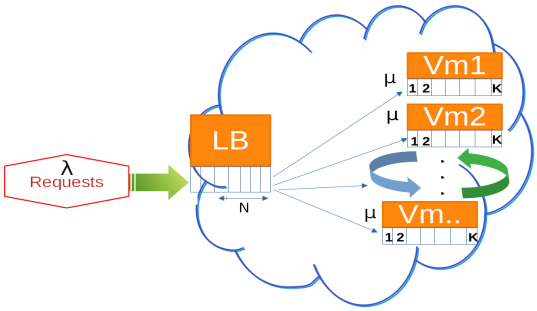
In this paper, we limit ourselves to cloud systems that allow only the STR auto-scaling method. The decision to scale out is taken when the load (the number of request) on the dispatcher is greater than or equal to N, and the decision to scale in is taken when a started resource (a virtual machine for horizontal auto-scaling, a Core or CPU for vertical auto-scaling) is idle for **α** seconds. Our goal is to determine the optimal couple (N, **α**) that minimizes the used resources without violating SLA in horizontal auto-scaling model and also in vertical auto-scaling model. We propose modeling the auto-scaling mechanism by the Stochastic Well-formed coloured Nets (SWN). We conduct simulation of the models to determine the best couple (N,**α**) for each one. Our model takes as input the arrival rate, the service time rate, the queue capacities, the SLA. The latter is defined in this work by mean response time.

The rest of the paper is organized as follows: the following section provides a description for the systems being modeled. The third section outlines the objective of the work. The fourth section presents the SWN models. The fifth presents the results of the simulation. The last section concludes the paper and gives some perspectives.

DESCRIPTION OF MODELED SYSTEMS

In order to model auto-scaling mechanism in Cloud Computing, we have considered the architecture illustrated on Figure 1 and described as follows: we have a system consisting of a set of servers (virtual machines: VM_1, VM_2, ...) and a Load Balancer (LB). These servers are identical and run the same stateless application. Each server has a service rate μ. The LB controls and manages the set of servers.

The requests arrive on LB with an arrival rate λ. The later forward them to servers for execution. Each server has a local queue of capacity K that stores requests waiting for their execution. The LB also ensures the admission control and the auto-scaling mechanisms. It has a queue with infinite capacity. In terms of admission control, when the LB receives a new request, it evaluates the load of all active servers and determines the server that should receive the request (using a certain policy). In case of saturation of all servers, the LB stores the request in its queue. When the queue size of the LB reaches a length N (a predefined parameter), the auto-scaling mechanism increases resources by adding a server in the case of horizontal auto-scaling or by adding a Core in the case of vertical auto-scaling. After this operation, the requests at the LB will be distributed with the new capacities or speeds of the system obtained from auto-scaling mechanism.



Figures 1: Architecture of the modeled system

In this system, when the horizontal auto-scaling mechanism is used, a server that remains idle for **α** (a predefined parameter) seconds is turned off, and in case of vertical auto-scaling, a core that remains idle for **α** seconds is removed.

OBJECTIVE

In this work, we sought to determine the best system states to operate the auto-scaling mechanism. The system state is defined by the number n of waiting requests on LB queue, the vector $\boldsymbol{v}=(x_1, x_2,\ldots,x_m)$ that contains the idle times of all started resources. x_i is the idle time for the resource i; $x_i=0$ if the resource i is executing requests; $x_i=s$ if the resource i is free since s second. If we observe a system state for which $n=N$ then we immediately add a new resource to the system to increase its capacity. If we observe a state of the system for which a value of $x_i=\boldsymbol{\alpha}$ then resource i is immediately switched off to decrease the capacity of the system.

The N and **α** that we are looking for are those that maximize the utilization rate of resources allocated to the application, minimize the amount of resources used by the application, while satisfying the level of service requested by the user through the SLA.

In this paper, the SLA is defined by mean response time. To find the best N and **α**, we conduct simulation of the model, monitor the mean response time and the amount of unused resources. Indeed, in the event of an overestimation, the indicator of unused capacity shows the low percentage of resource utilization. In the event of an underestimation, the response time indicators will show us high response time. These indicators, gathered together, can be used to fine the best N and **α** for an auto-scaling approach adapted to the context of the application concerned.

SWN MODELING FOR AUTO-SCALING

In this paper, we model the auto-scaling mechanism by SWN (Chiola et al. 1993). The SWN is an extension of colored petri nets. The main interest of this model is the possibility to have a reduction due to the symmetries of the Markov chain derived from the stochastic colored Petri nets (Chiola et al. 1993; Haddad and Moreaux 2009). In a SWN model the tokens have a color of a given set

which allows to model the characteristics of different entities of the same type (for example different types of servers or different types of requests); Places and transitions have a color domain. A color domain of a place identifies the tokens it can contain; that of a transition defines the type of values used to make it enabled. A color domain is a finished Cartesian product of elementary classes. Each elementary color class is a finite, non-empty set of terminal colors whose definition does not depend on any other color.

The other interesting aspect of SWN for our work here is the implicit synchronization between tokens of the same class. This allows us to easily define synchronizations based on the membership of a token or the membership of a part of a token. For example, when we are going to dispatch requests in the queues of different servers, we are going to do it associating for each request the name of the server that is going to execute it and there the implicit SWN synchronization guarantees that this request will only be executed by a CPU or Core of the designated server.

In this work, we are interested in horizontal auto-scaling and vertical auto-scaling. We propose a SWN model for each of them.

The SWN Model for Horizontal Auto-scaling

The Horizontal Auto-scaling SWN model that we propose is shown in Figure 2. For this model, we considered two types of servers: permanent servers and dynamic servers. Permanent servers refer to virtual machines that are started from the beginning and kept running all time. They represent the initial capacity of the system. The dynamic servers, on the other hand, are the virtual machines that will be created and later stopped, if necessary, by the auto-scaling mechanism.

We have two color classes to model the servers in our SWN model: the "PSRV" class represents permanent servers and the "DSRV" class represents dynamic servers. The "PSRV" class is used to initialize the marking of the "RunningVM" place, the "DSRV" class initialize marking of the "MaxCapa" place which represents the maximum capacity that auto-scaling can allocate to our system. The "RunningVM" place gives the number of servers running at a given time.

The requests arrive on the Load Balancer (the "LB" place) through the "IncomingRequests" transition. The "IncomingRequests" transition models the incoming flow of requests with arrival rate **λ**. The immediate transition "Affect" expresses the random dispatching of these requests to virtual machines. This dispatching is conditioned by the existence of free capacity on the servers.

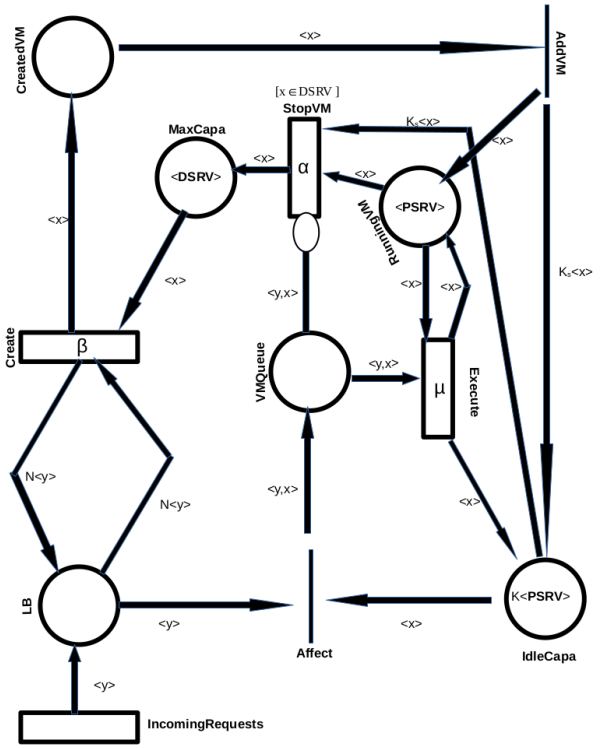


Figure 2: The SWN model for horizontal auto-scaling

The "IdleCapa" place models the available capacity in the system at a given time. The initial marking M_0 of this place is the length of all queues of all permanent servers.

$$M_0(\text{IdleCapa}) = K\langle PSRV\rangle$$

where <PSRV> gives the number of permanent servers. The "VMQueue" place represents the queues of all running servers. Each token in this place is <y, x> tuple composed of a request "y" and the name of the virtual machine "x" that will execute this request. The "Execute" transition models execution of requests and is defined using the **μ** service rate. This transition is configured in "infinite server" mode to model the parallel execution of virtual machines.

In this model, the auto-scaling mechanism is modeled by the "Create" and "StopVM" transitions. The "Create" transition models the creation of a new VM by the auto-scaling mechanism. The "StopVM" transition models the release of a VM that has no more requests to execute. For each of the two transitions, we have an exponential distribution to model the time required to complete the task in question. For the "Create" transition, the rate is $1/\beta$ where β is average time needed to create a new VM. For the "StopVM" transition, the rate is $1/\alpha$ where α is the average time to wait before stopping an idle VM.

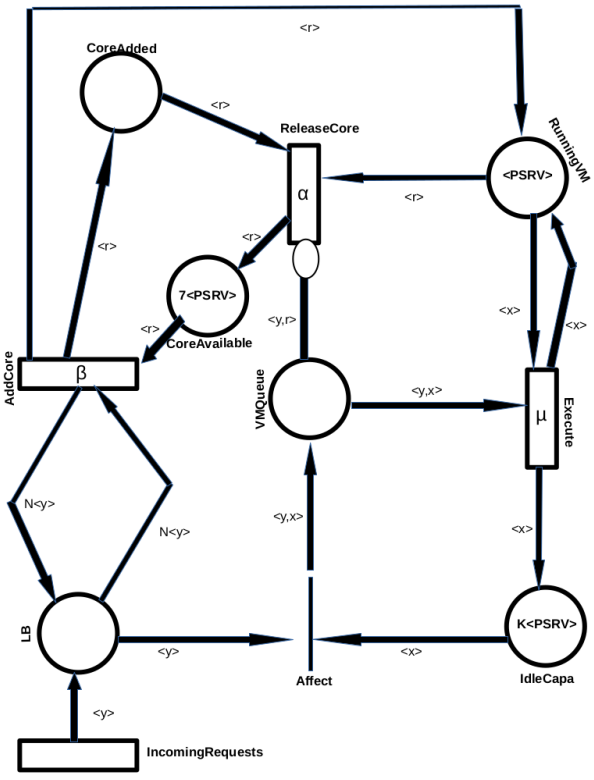
The "Create" transition requires a number N of tokens to create a new VM. This parameter "N" corresponds to the number of requests exceeding the capacity of the system and from which the scale out is operated. The

worst case with this parameter N is N=1 which means that a new VM is added as soon as there is a single request exceeding the system capacity. Indeed, delaying the addition of new resources pushes the system to make better use of the resources already allocated, hence the idea of looking for the largest N that respects the QoS.

The defined guard on "StopVM" transition indicates that only dynamic servers can be shut down. Stopping a VM causes the disappearance of "K" tokens from the "IdleCapa" place. This disappearance expresses the removal of its queue from the overall system capacity. The "AddVM" transition adds "K" tokens into the "IdleCapa" place when a new VM is created with the auto-scaling mechanism. The addition of "K" tokens into the "IdleCapa" place represents the creation of a new queue for the newly created VM.

The SWN Model for Vertical Auto-scaling

Figure 3 gives vertical auto-scaling SWN model. Here, we only have permanent servers. The computing capacity of these servers can be increased by adding new cores. The "CoreAvailable" place contains the number of cores available on the physical machines hosting the virtual servers. A core can be added to one of the running servers, if waiting requests on LB is equal to N.



Figures 3: The SWN model for vertical auto-scaling

The time required to add a Core is an exponential with rate **1/β** represented by "AddCore" transition. The "CoreAdded" place helps to recognize the Cores added by the auto-scaling mechanism. The "ReleaseCore" transition rate defines the time **α** after which an unused Core is released. The other elements of the model play the same roles as in the horizontal model (see Figures 3).

SIMULATION AND ANALYSIS OF RESULTS

Performance Measures

In this work, we use the following performances measures:

1. Average Response Time of the system (ART);
2. Average number of Resources Used (ARU).
3. Resource Utilization Rate (RUR)

These performance are calculated as follows:

$$\text{ART}=\frac{E(\text{LB})+\text{E}(\text{VMQueue})}{X(\text{IncomingRequests})}$$

$$\text{ARU}=\text{E}(\text{RunningVM})$$

where "E" is a function of GreatSPN(Amparore et al. 2016) that gives the average number of tokens of the place whose name is passed as an argument; and "X" is the function of GreatSPN that gives the average throughput of the transition whose name is passed as an argument.

$$\text{RUR}=100-\text{IC}$$

where

$$\text{IC}=\begin{cases}\text{AFP}-\text{ARS} & \text{IfAFP}\geq\text{ARS}\\ 0 & \text{if AFP}<\text{ARS}\end{cases}$$

where ARS is the average number of requests stored in the load balancer's queue.

$$\text{ARS}=\frac{E(\text{LB})*100}{E(\text{RunningVM})*K}$$

and AFP is the average number of free places in server queues:

$$\text{AFP}=\frac{E(\text{IdleCapa})*100}{E(\text{RunningVM})*K}$$

Simulated Examples

To analyze these models, we used GreatSPN's simulator (Amparore et al. 2016). The servers on which the application is running have each one a service rate μ = *4 requests per time unit.* In this work, we take the second as the basic unit of time for our model. We have defined our basic configuration with two permanent servers. The maximum number of extra resources that auto-scaling mechanism can create for our system is limited to 10 VMs in horizontal case, and 14 Cores in vertical case. The queue capacity K of a VM is equal to

50. For the sake of simplicity and without loss of generality, we will assume that all flavours are single-core in the case of Horizontal Auto-scaling. The creation time of a new resource is an exponential distribution with rate 1/β=0.5. This means that it will take 2s on average to create a new VM or add a Core in the vertical case. The incoming flow rate **λ**=7. The QoS constraint to be guaranteed is the "average response time" (ART), and it must remain ≤ 7second. Table 1 summarizes the different parameters used in our simulation.

Table 1: Simulation parameters

Parameter	**Values**
Mean response time	≤ 7s
Service rate (***μ***)	4 requests per second
VM's Queue capacity (*Ks*)	50
LB's queue capacity (K_L)	∞
Permanent servers	2 VM
Number of extra resources	10 VMs or 14 Cores
Arrival rate *λ*	7
Rate *1/β*	0.5
Rate *1/α*	{0.1, 0.5}
N	{1, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 120}

In order to determine the best couple (*N*,*α*), we simulate the models and calculated the performance measures for several *N* values and for several *α* values, see Table 1.

To quickly find the best couple (*N*,*α*), we can begin research with large *N* values and small values of *α*. For example we can start with *N*=*K* and depending on the average response time observed and its acceptable limit value, we will know if it is necessary to increment or decrement the *N*. However, if we want to show the general behavior of Auto-scaling mechanism, we can begin with the worst value *N*=1. Here, we chose to present simulation result with the last case. The increment step to reach the optimal value is 10.

Table 2 shows the average response time (ART), the resource utilization rate (RUR), the average number of VMs (ARU) used for horizontal auto-scaling as a function of N and *α*. We observe that small values of N give small ART but lead to low resource utilization rates. The increase of N increases the RUR; that is good things but increase also the ART. The increase of the latter is blocked by constraint of the SLA. So we see clearly with the constraint on the SLA (ART must be inferior or equal to 7 second), that the maximum resource utilization rate is 77.52% with N=70. We simulate the system with several value of *α* varying from 2 to 10 per step 2, but in Table 2, we report result for only *α* =2 and *α* =10. Comparing performance for *α*=2 and *α*=10, we observe that *α*=2 give better rate of utilization resource.

Table 2: System performances as function of N and *α* with an horizontal auto-scaling mechanism

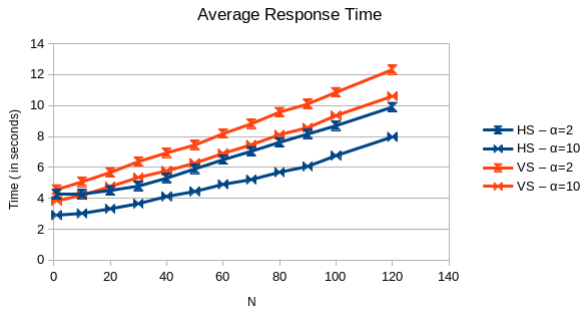
H-Scaling	α=2			α=10		
N	ART	RUR	ARU	ART	RUR	ARU
1	4.25	45.85	2.6	2.91	29.28	2.78
10	4.28	46.46	2.59	3.02	30.52	2.77
20	4.51	48.52	2.61	3.32	33.88	2.76
30	4.78	51.68	2.6	3.65	37.42	2.74
40	5.3	57.58	2.58	4.11	42.5	2.7
50	5.9	64.39	2.56	4.44	45.96	2.71
60	6.49	71.47	2.54	4.9	51.2	2.69
70	7.04	77.51	2.54	5.21	54.4	2.68
80	7.62	84.57	2.52	5.68	59.71	2.67
90	8.15	91.11	2.51	6.06	63.62	2.66
100	8.69	96.64	2.52	6.76	71.93	2.64
120	9.9	100	2.49	7.97	85.72	2.61

Table 3 shows the average response time (ART), the resource utilization rate (RUR), the average number of Cores (ARU) used for vertical auto-scaling as a function of N and *α*. We observe the same result as in auto-scaling horizontal.

Table 3: System performances as function of N and *α* with an vertical auto-scaling mechanism

V-Scaling	α=2			α=10		
N	ART	RUR	ARU	ART	RUR	ARU
1	4.56	71.71	2.56	3.83	68.24	2.56
10	5.06	76.82	2.52	4.2	70.81	2.52
20	5.67	83.62	2.52	4.75	75.97	2.52
30	6.37	91.39	2.5	5.34	81.8	2.5
40	6.94	97.56	2.46	5.78	86.04	2.46
50	7.44	100	2.47	6.27	91.02	2.47
60	8.18	100	2.45	6.9	97.5	2.45
70	8.82	100	2.47	7.46	100	2.47
80	9.57	100	2.43	8.12	100	2.43
90	10.1	100	2.43	8.57	100	2.43
100	10.85	100	2.45	9.35	100	2.45
120	12.32	100	2.43	10.6	100	2.43

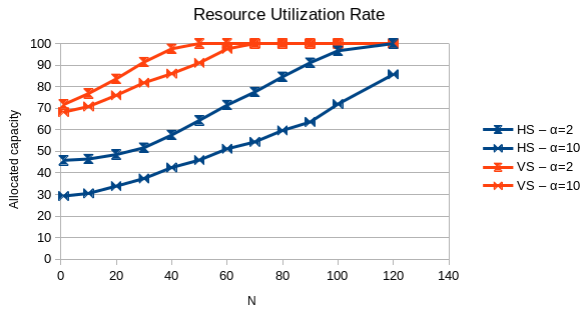
Figure 4 shows the evolution of the mean response time for the two types of auto-scaling and for two α values.



Figures 4: Average Response Time as function of N

These results show that delaying the addition of resources leads to an increase in the mean response time for all types of auto-scaling. They also show that delaying the release of inactive resources improves the mean response time.

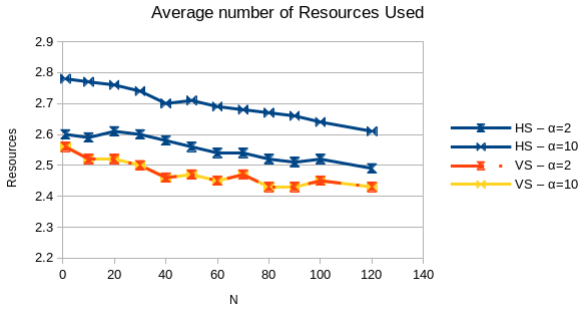
It can also be seen that the average response time recorded by the Horizontal Auto-scaling in this specific case is smaller than the average recorded with the Vertical Auto-scaling mechanism. However, in terms of resource utilization rates, the results in Figure 5 show a higher utilization rate with Vertical Auto-scaling compared to Horizontal Auto-scaling.



Figures 5: Resource Utilization Rate as function of N

We observe in Figure 5 that delaying scale-out increases the rate of resource utilization. It can also be seen that delaying the release of resources naturally leads to a significant waste of resources, particularly for the horizontal mechanism.

The same reading can be made from Figure 6. Here, the vertical auto-scaling recorded a better average of Cores used. In addition we can also see that this score remained constant with the two **α** values.



Figures 6: Average number of Resources Used as function of N

According to these results and in order not to violate the fixed QoS constraint, the N to be retained can be chosen among the following values in Table 4. The best couples were highlighted.

Table 4: The best couples

	Candidate couples	Recorded utilization rate	Mean number of Cores used
Horizontal scaling	**N=70,α=2**	77.52%	2.54
	N=100,α=10	71.93%	2.64
Vertical scaling	N=40, α =2	97.56%	2.46
	N=60, α =10	97.50%	2.45

CONCLUSION AND PERSPECTIVES

In this paper, we proposed a modeling of the auto-scaling mechanism with SWN models. We studied two types of systems. The first system uses the horizontal auto-scaling mechanism, and the second system uses the vertical auto-scaling mechanism. For each system, the arrivals requests follow a Poisson process, and the service times follow an exponential distribution.

The simulation of models allowed us to fine the best load N of LB from which we have to scale out, and the best idle time α of a resource from which we have to scale in. The best couple (N, α) allows us to minimize the number of resources to be allocated to the system on the one hand and on the other hand to ensure a high rate of use of allocated resources. In a future work, we want to test our using real data. It would also be interesting to study self-scaling mechanisms with time-varying arrival rates.

REFERENCES

Al-Dhuraibi Y., Paraiso F., Djarallah N., Merle P.. Elasticity in Cloud Computing: State of the Art and Research Challenges. *IEEE Transactions on Services Computing, IEEE*, 2018, 11 (2), pp.430-447. ⟨https://dx.doi.org/10.1109/TSC.2017.2711009⟩.

Amparore E.G., Balbo G., Beccuti M., Donatelli S., Franceschinis G. (2016) 30 Years of GreatSPN. In: Fiondella L., Puliafito A. (eds) Principles of Performance and Reliability Modeling and Evaluation. *Springer Series in Reliability Engineering. Springer*, Cham. https://doi.org/10.1007/978-3-319-30599-8_9

Chiola G., Dutheillet C., Franceschinis G. and Haddad S., "Stochastic well-formed colored nets and symmetric modeling applications," in *IEEE Transactions on Computers*, vol. 42, no. 11, pp. 1343-1360, Nov. 1993, https://doi.org/10.1109/12.247838

Fallah, M., & Arani, M.G. (2015). ASTAW: Auto-Scaling Threshold-based Approach for Web Application in Cloud Computing Environment. *International Journal of u- and e- Service, Science and Technology*, 8, 221-230.

Haddad S., Moreaux P., Stochastic Well-formed Petri Nets. *M. Diaz. Petri Nets: Fundamental Models, Verification and Applications, Wiley-ISTE*, pp.303-320, 2009. ⟨http://hal.univ-smb.fr/hal-00441928⟩.

Han R., Guo L., Ghanem M., and Guo Y., "Lightweight Resource Scaling for Cloud Applications," in *2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, May 2012, pp. 644–651, (https://doi.org/10.1109/CCGrid.2012.52).

Hector F., Guillaume P., and Thilo K.. 2014. Autoscaling Web Applications in Heterogeneous Cloud Infrastructures. In *Proceedings of the 2014 IEEE International Conference on Cloud Engineering (IC2E '14). IEEE Computer Society*, USA, 195–204. DOI:https://doi.org/10.1109/IC2E.2014.25.

Kumar J., and Singh A. K. 2018. Workload prediction in cloud using artificial neural network and adaptive differential evolution. *Future Gener. Comput. Syst. 81, C* (April 2018), 41–52. DOI:https://doi.org/10.1016/j.future.2017.10.047

Qu C., Calheiros R. N., Buyya R., 2016. A reliable and cost-efficient auto-scaling system for web applications using heterogeneous spot instances. *J. Netw. Comput. Appl. 65, C* (April 2016), 167–180. DOI:https://doi.org/10.1016/j.jnca.2016.03.001

Roy, N., Dubey, A., and Gokhale, A. 2011. Efficient Autoscaling in the Cloud Using Predictive Models for Workload Forecasting. In Proceedings of the *2011 IEEE 4th International Conference on Cloud Computing (CLOUD '11). IEEE Computer Society*, USA, 500–507. DOI:https://doi.org/10.1109/CLOUD.2011.42

Sharma U., Shenoy P., Sahu S. and Shaikh A., "A Cost-Aware Elasticity Provisioning System for the Cloud," *2011 31st International Conference on Distributed Computing Systems*, Minneapolis, MN, USA, 2011, pp. 559-570, https://doi.org/10.1109/ICDCS.2011.59.

Singh P, Gupta P, Jyoti K and Nayyar A (2019). Research on Auto-Scaling of Web Applications in Cloud: Survey, Trends and Future Directions. *Scalable Computing: Practice and Experience*, 20(2), 399–432.

Tesauro, G., Jong, N. K., Das, R., and Bennani, M. N. 2006. A Hybrid Reinforcement Learning Approach to Autonomic Resource Allocation. In Proceedings of the *2006 IEEE International Conference on Autonomic Computing (ICAC '06). IEEE Computer Society*, USA, 65–73. DOI:https://doi.org/10.1109/ICAC.2006.1662383

Villela, D., Pradhan, P., and Rubenstein, D. 2007. Provisioning servers in the application tier for e-commerce systems. *ACM Trans. Internet Technol.* 7, 1 (February 2007), 7–es. DOI:https://doi.org/10.1145/1189740.1189747

AUTHOR BIOGRAPHIES

Mohamed M. O. DEYE is an Assistant Professor at the Department of mathematics and computer science at Cheikh Anta Diop University of Dakar, Senegal. His areas of interest are Cloud Computing, Web Services and performance evaluation of distributed systems. His email address is mohamed.oulddeye@ucad.edu.sn

Mamadou THIONGANE is an Assistant Professor at Cheikh Anta Diop University of Dakar, Senegal. . His main research interests are waiting time prediction in call centers, modeling service time in service system. He are also interest by Cloud Computing system. His email address is mamadou.thiongane@ucad.edu.sn

MBAYE SENE is a full professor in the department of Mathematics and Informatics of the Faculty of Sciences and Technology in the university of Cheikh Anta Diop of Dakar. He received the PhD degree in Computer Science from the University of Paris Dauphine, France, in 2002 and a master degree of Management of organizations in the same university in 2003. He was during 2 years an associate professor in Paris-Dauphine before he integrates the most important university in Senegal, UCAD. His main research interests include distributed database systems, design of inter-operate open systems, wireless de sensor networks and performance evaluation of stochastic complex systems. He has authored or co-authored more than 20 international conference papers and journals. Professor Mbaye SENE mange also since 2008 big projects in the ministry of employment and vocational training of Senegal; he has worked with France Agency of Development (AFD), GIP International (France).