A Workload-adaptable architecture for migrated software to the Cloud

Zahra Reisi[1]*, Omid Bushehrian[2], Farahnaz RezaeianZadeh[3]

[1,2] *Department of Computer Engineering and IT, Shiraz University of Technology, Shiraz, Iran*

[3] *Department of Computer and Information Technology, Foolad Institute of Technology, Isfahan, Iran*

Abstract

Due to the elastic nature of the cloud environments, migration of the legacy software systems to the cloud has become a very attractive solution for service providers. To provide Software-as-a-Service (SaaS), an application provider has to migrate his software to the cloud infrastructure. The most challenging issue in the migration process is minimizing the cloud infrastructure (VM's) costs while preserving the quality requirements of the service consumers. In this paper a self-adaptive architecture for migrated applications to the cloud is proposed in which an intelligent auto-scaling component continuously monitors the incoming application workload and performs vertical or horizontal scaling. Since reacting to the transient workload changes always results in useless sequences of "acquire-release" actions of cloud resources and imposes unwanted overhead costs on the service provider, the auto-scaling component recognizes the transient workloads using a Learning Automata and only reacts to the stable ones. The OpenStack platform is used for evaluating the applicability of proposed architecture in real cloud environments. The experimental results demonstrate the ability of the proposed approach in recognizing the transient workloads and consequently reducing the overall costs of the service provider.

Keywords:*Auto-Scaling,Cloud Computing, LearningAutomata, Transient Workload,Virtualization Manager.*

1. Introduction

The SLA's (Service Level Agreements) between service provider and service consumers are the most important artifact upon which the required resource capacities of the migrated application to the cloud areplanned. However the fluctuation workload of recent cloud applications requires an elastic mechanism to adapt the amount of the acquired resources to the current workload [1, 2]. It is essential for the cloud provider to minimizing the cost of rented resources while preserving the QoS requirements specified in the SLA's during the application execution. To achieve this, an important task is to find the optimal amount of resources for different kind of workloads before migrating the application to the cloud which is performed during the capacity planning activity. Usually the service provider scaling is driven by a set of scaling rules which specify the correct time for scaling up or down [3, 4].

With APIs provided by commercial cloud providers such as Amazon, the developer can define and configure the auto-scaling rules. In order to scaling up/down the Amazon VM's, an object called*Amazon Cloud Watch Alarm* should be applied. This object is a monitor agent responsible for checking a VM QoS status over a period of time and triggering the scaling rules when the QoS falls below a predefined threshold. The scaling rules are defined for changing (add/delete) the number of VM instances (horizontal scaling) or changing the size of current VMs (vertical scaling).

However an important open issue regarding the auto-scaling mechanisms is the adaptation of scaling rules based on runtime conditions: The scaling rule set may involve some probabilistic rules whose triggering probabilities change over time based on the stability or instability of the application workloads. For instance consider scaling rule saying that "if the workload changes to L_i then add one VM to current VM group". However if the workload L_i is not stable and immediately vanishes, changing the current VM set and software deployment is useless. Generally, reacting to the transient workload changes always results in useless sequences of "acquire-release" actions of cloud resources which

impose unwanted overhead costs on the service provider due to the configuration changes. Therefore it is essential to foresee the future conditions of the system workload by learning from previous events.
In this paper, a new self-adaptive architecture for migrated applications to the cloud is proposed. The main objective of our strategy has been to predict the transient workloads and react intelligently to them. We believe that by this strategy, the total resource cost of the service provider is reduced significantly as the reconfiguration process happens only when the change is stable. We have implemented the proposed approaching the OpenStack environment [5].
The paper is organized as follows: section 2 includes a review of the literature on auto-scaling in the cloud. Our strategy is introduced in section 3 and the experiment result is demonstrated in section 4. The conclusion and the future work is described in section 5.

2. Related works

Workload prediction is the most applied technique for auto-scaling in cloud environments. The main objective here is to address the challenges of horizontal and vertical scaling. A framework is proposed in [6] for automatic scaling called ''SmartScale'' in which the horizontal or vertical scaling is applied at each stage to make a trade-off between cost changing configuration and SLA violation. *SmartScale* uses horizontal and vertical scaling together and in correct time by means of the decision tree technique. The scaling process is executed when the workload is changed. In [7], a controller is embedded within the auto-scaling component for capacity planning to find the optimal number of resourcescorresponding to the upcoming workload. In [8] a linear regression with an auto-correlation function is used for predicting the number of web requests. The main objective of this mode is to find an optimal trade-off between cost and latency when resources are allocated. The relationship between the number of VMs and the latency is defined based on an M/M/m queuing model. In [9] an efficient model is proposed for resource allocation based on the prediction techniques. Hereby using the second order Auto Regression Moving Average method (ARMA) it was shown that using small number of VMs can effectively satisfy QoS requirements and also cost reduction. In [10] the scaling problem is formulated as an integer programming problem. The architecture for scaling has two parts including runtime and pre-runtime. The results showed an optimal trade-off between cost and SLA objectives.
In some studies on the transient and stable workloads, the stableness of the workload is defined statically [11-12]. However, it is highly dependent to workload changes. To address this shortcoming, in [13] a dynamic threshold based on meta-rules is proposed by which the threshold is adapted based on the workload changes.
An important ignored aspect in the most of previous studies is the effect of transient workload on the cloud cost which is paid by the service provider. We addressed this shortcoming in our previous paper in [15] where an intelligent Auto-Scaling Engine (iScale) is presented which recognizes the transient workload changes using a Learning Automata and only reacts to the stable workloads.
In this paper the architecture presented in [15] is improved by augmenting a new component called "Virtualization manager" which is responsible for interfacing with cloud virtual machine manager (hypervisor) to perform vertical or horizontal scaling, Moreover the learning method is enhanced in comparison to our previous study.

3. Cloud Auto-Scaling Approach

The proposed auto-scaling approach applies both vertical scaling and horizontal scaling. Auto-scaling is performed based on the type of workload. There are two types of workload: stable load and transient load. Identifying the type of workload is performed by learning automata. The proposed method in this paper satisfies three important requirements: reducing the cost of scaling, fulfilling the SLA constraints, and decreasing the number of runtime software re-configuration.
There are durability thresholds corresponding to each workload computed at design time considering the cloud costs and SLA violation penalty costs by which the durability of a given workload is determined at runtime. If a given workload is not changed before its corresponding threshold, it is recognized as a "stable load" otherwise it is a "transient load". The durability of each load change from L_i to L_j is learned by a Learning Automata LA_{ij} based on its durability history. If a load change from L_i to L_j is learned as "stable" by LA_{ij}, it means that the act of scaling out/in the current configuration to the new configuration $C_j$is more probable than keeping the current configuration.
Assuming that the migrated software initially is deployed on a set of virtual machines acquired from the IaaS provider, scaling up (or down) the system based on changing the workload means that the allocated resources to the software and the assignment of software components to the virtual machines may change and a new configuration is installed for the software. Prior to the migration process, the required configuration G_i to support the known workload L_i has to be specified by the designer during an activity called Capacity Planning[14]. Here, based on the QoS requirements of the end-users specified in the Service Level Agreements (SLA), the designer determines the required capacity (the number of virtual machines and the size of allocated resources on each one) and the optimal assignment of software components to virtual machines such that the SLA constraints are not violated for each known workload L_i. The outcome of this activity is the Configuration Plan of the migrated software which is defined as follows [15]:

Definition 3.1:A function which associates each workload L_i with its corresponding configuration G_i is a Configuration Plan P such that: 1) none of the SLA constraints are violated when the system workload is L_i and the software configuration is G_i. 2) In order to satisfy the first condition G_i contains the least resources.
Definition 3.2: The Configuration G_i (defined in definition 3.1) corresponding to load L_i is defined as triple (C, M, A) where C is the set of software components to be started during the execution of the software with this configuration, M is the set of VM's on which the software components are deployed and A is a function that maps each component $c_k \in C$ to a $vm_j \in M$.

3.1.Architecture

The proposed self-adaptable architecture is shown in Fig.1 a*configuration* contains a set of virtual machines allocated to the application components based on the Configuration Plan. *Load Watch* monitors the incoming traffic of virtual machines. iScale component [15] uses a load Watch to identify the current workload classes and classifies the workload into one set of pre-created workload classes then sends it to the learning automata. The iScale uses the average of time interval between two requests to classify the workload [15]. *Learning Automata* decides when and how to scale the cloud service. In the real-time scaling, the learning automata component proposes a scaling strategy according to its learning for transient workload. *Configuration Plan* represents the current status configuration.

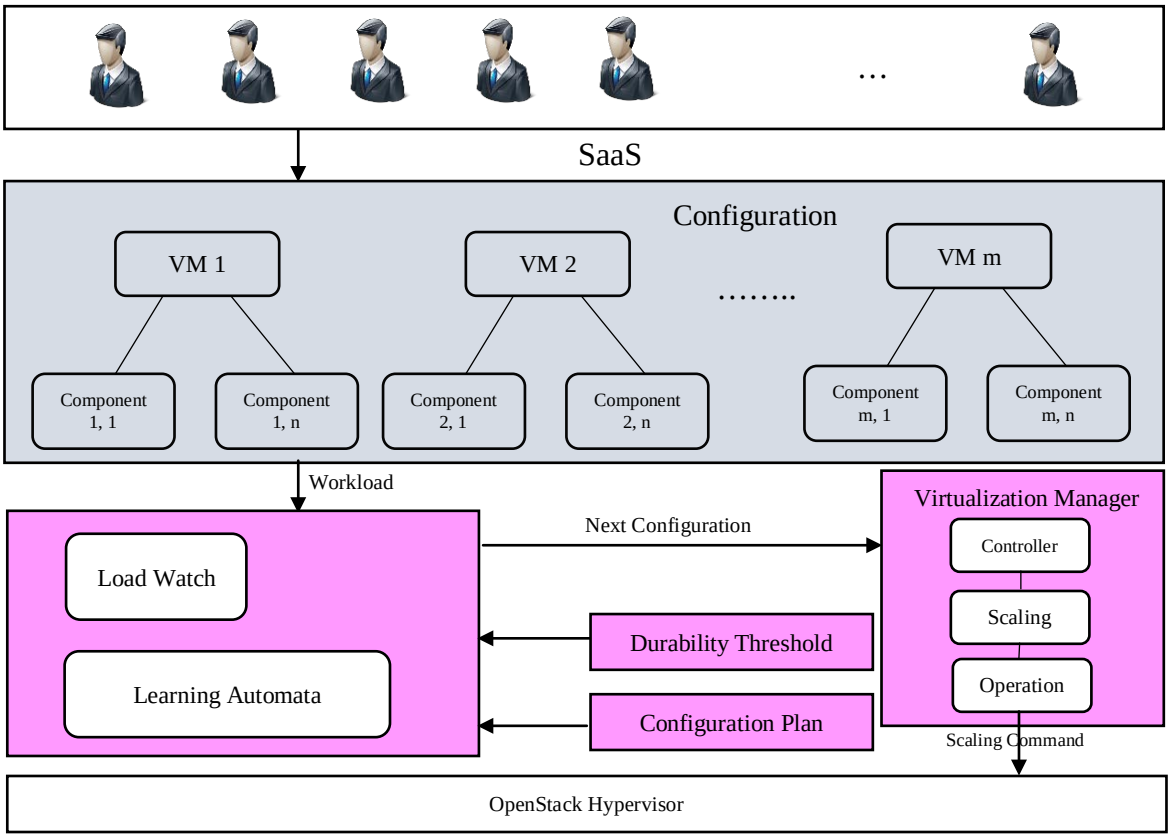

Fig. 1: The role of **"**Virtualization Manager"component inthe proposed Architecture

3.2. Virtualization Manager

Scaling commands are executed by the virtualization manager. Virtualization Manager receives the new configuration and uses VM resizing toscale up/down the current configuration. Fig. 2 represents the relationship between Virtualization Manager Section and the main APIs provided by the cloud hypervisor.
During the learning process, when a load change happens, the scaling engine may falsely keep the current configuration due to the recognition of the newly workload as transient. To address this problem, the controller component is used in our proposed method to monitor and correct the wrong decisions. Adding or releasing the resources is also done by the virtualization manager.

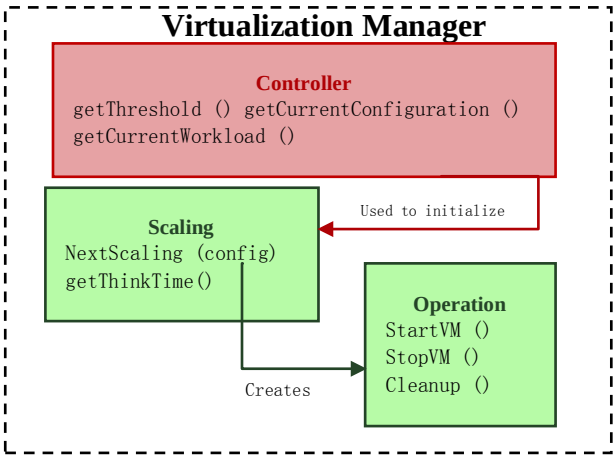

Fig.2: Virtualization manager APIs

3.3. Cost model and Parameters

Suppose the input workload is L_c and the current configuration of the application is C_i, if the workload changes to the other class such as L_j, may be reacted by the auto-scaling engine either by scaling out the configuration C_i and installing the new configuration C_j or ignoring the load change and continue with the current configuration. If the first action is taken, a vertical or horizontal scaling may happen, in which the cost of scaling after t seconds is calculated as follows [15].

$$ScalingCost_{Horizontal}(t, i, j) = \int_0^{t_{c_{ij}}} P_{change}(t) + Cost_{Cj} \times (t - t_{c_{ij}}) + SoftwareLicenceCost_{i,j} \quad (1)$$

$$ScalingCost_{Vertical}(i, j) = Cost_{c_j} \times \alpha_{ij} \quad (2)$$

Where $t_{c_{ij}}$ denotes the reconfiguration time from C_i to C_j, $P_{change}(t)$is the SLA violation function related to the horizontal configuration, $Cost_{Cj}$ is cost of new configuration,α_{ij} denotes the durability threshold regarding the current configuration C_i and new load L_j and $SoftwareLicenceCost_{i,j}$ is the cost of new software licenses in configuration C_j.
If the action is the one that ignores the workload change, the current configuration is kept and the total cost incurred at time t is calculated as follows:

$$IgnoringCost(t, i, j) = \int_0^t P_{c_i l_j}(t) + Cost_{Ci} \times t \quad (3)$$

The penalty function due to violation of SLA is denoted by$P_{c_i l_j}(t)$, the cost of the current configuration is shown by $Cost_{Ci}$.
The threshold valueα_{ij} is the time point at which the cost of changing the configuration calculated by formula (1) or (2), and the cost of keeping the current configuration calculated by formula (3) are equal. During the workload change, as the SLA violation is increased gradually, the cost of keeping the current configuration is always less than the cost of re-configuration while the threshold time α_{ij} is not reached. This reasoning is obvious due to the fact that function $P_{c_i l_j}(t)$ assumed as a Strictly Increasing Function.

3.4. Learning Method

Learning Automata is a finite state machine that aims to apply the best action on environment through a learning process. The best action is the one that maximizes the probability of receiving rewards from the environment. LA chooses an action repeatedly based on the action probabilities and then updates the action probabilities considering the environment responses.

Learning automata can be presented formally as the tuple: $(x, \emptyset, \alpha, P, A, G)$,where x is the input set, $\alpha = \{\alpha_1, \ldots, \alpha_r\}$ is a set of automata actions, $F = \emptyset \times \beta \rightarrow \emptyset$is the production function which determines the new automata state based on the current state and the inputs, $P(n)=\{P_1(n), \ldots, P_r(n)\}$ denotes the action probabilities at stage n, $G = \emptyset \rightarrow \alpha$ a function that maps the current state into the current output, $\emptyset = \{\emptyset_1, \emptyset_2, \ldots, \emptyset_k\}$ is the set of internal states and A is the learning algorithm [16].
In the proposed method, to learn the durability of each individual load change, correspondingto each load change from L_i to L_j a learning automata LA_{ij} with two actions {ignore, scale} is defined. The learning algorithm works as follows: Once a load change happens, the actual durability of the previous load is compared with its threshold value, if greater; the probability of scale action of the corresponding LA_{ij} is increased; otherwise the ignore action is reinforced [15].
Suppose that *m* is the number of configurations and *n* is the number of workload class, *m×n×(n-1)* number is consider for LA in the proposed approach.

4. Experimental results

OpenStack is one the popular open source software packages in cloud computing domain. This software controls a set of computing, storage and network resources and provides a rich REST API to manage them [17][18]. The auto-scaling activity is done with Heat component which is compatible with AWS Cloud and contains a set of templates which are configured within the OpenStack.
Heat-API component sends the API request to Heat-Engine in order to be processed. The user puts all the information related to the application in a text file (compute instances, storage, floating IPs). Then, during the application execution, the demanded resources in the text file are provisioned by Heat whenever the scaling is requested. The environment of OpenStack is monitored by Ceilometer and collects the required information about the resources and system performance. Finally, a combination of Heat and Ceilometer activities will complete the scaling process [19, 20, 21].
The scaling process in Heat is depicted in Fig. 3 the API service controls an alarm lifecycle, the Compute agent gathers the statistical information and Alarm evaluator brings the alarm definition using the API Service. After delivering the alarm by Ceilometer, the Heat engine should start the scaling operations.

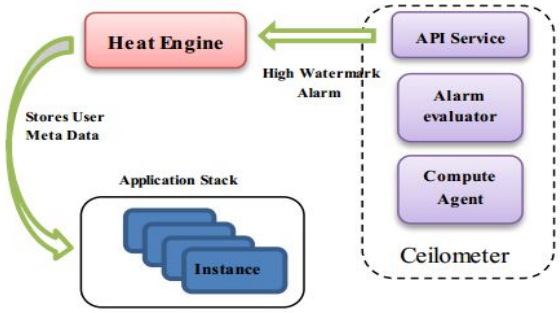

Fig. 3: Heat Engine scales out stack [22]

4.1.Experimental setup

The proposed approach is evaluated in single node OpenStack. The installation is done with Heat and Devstack. The experimental evaluation is designed to demonstrate the accuracy of our engine on enabling the cost-aware scaling in service clouds. The experiment is carried out on the OpenStack simulator which is a framework for modeling and simulating the cloud computing infrastructures and services. The price scheme is based on the Amazon EC2 pricing model [25]. Four types of VMs are provided in OpenStack: small, medium, large and xlarge, and they have different virtual resource costs, as shown in Table 1.

Table 1: The patch dimensions of the antenna

Configuration	Memory	CPUs	Disk	Price(per hour)
Small	2 Gb	1	20Gb	$0.026
Medium	4 Gb	2	40Gb	$0.116
Large	8 Gb	4	80Gb	$0.232
xLarge	16 Gb	8	160Gb	$0.420

Pre-defined classes are listed in Table 2. For each workload L_i a configuration G_i is defined. The workload used in this study is generated using the Rain tool [26]. Rain tool is a Markov-chain based workload generation toolkit which can easily use defined or empirical probability distributions to mimic different classes of load variations. We implemented the *Simple Scaling Method (SSM)* and the proposed scaling method (ASM). In the SSM re-configuration is performed after each workload change regardless of its durability.The first one will give a baseline for comparison purposes. Two types of composite workloads are used for experiments: Mixed and Transient.

Table 2: Different classes of workload

Workload Class	Mean time between requests(ms)
L_1	100
L_2	150
L_3	200
L_4	300

4.2.Evolutionresults

The effectiveness of our method is validated using the following experiments.

4.2.1. Experiment 1: workload type recognition

Here an experiment was designed to evaluate the performance of our method for recognizing the transient loads. Two modes of workloads are generated by means of Rain. As shown in Fig. 4 and Fig. 5, ASM could successfully recognize all the transient loads.

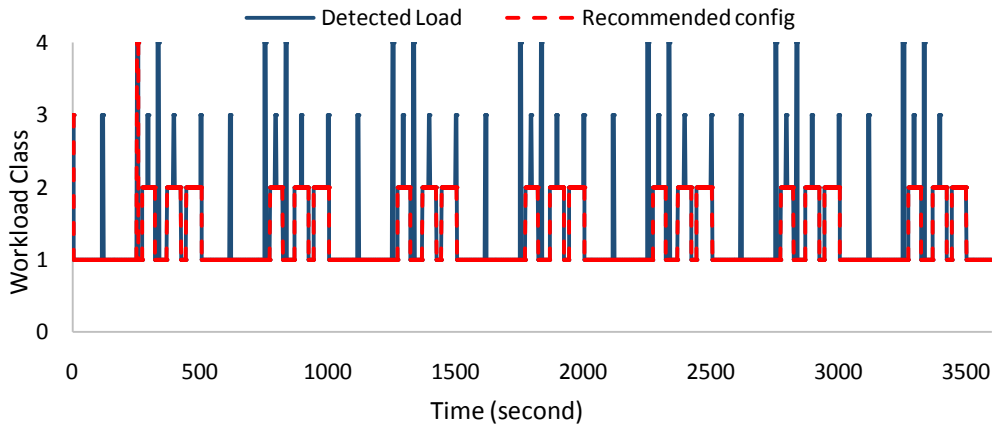

Fig. 4: detected transient workload in mixed status

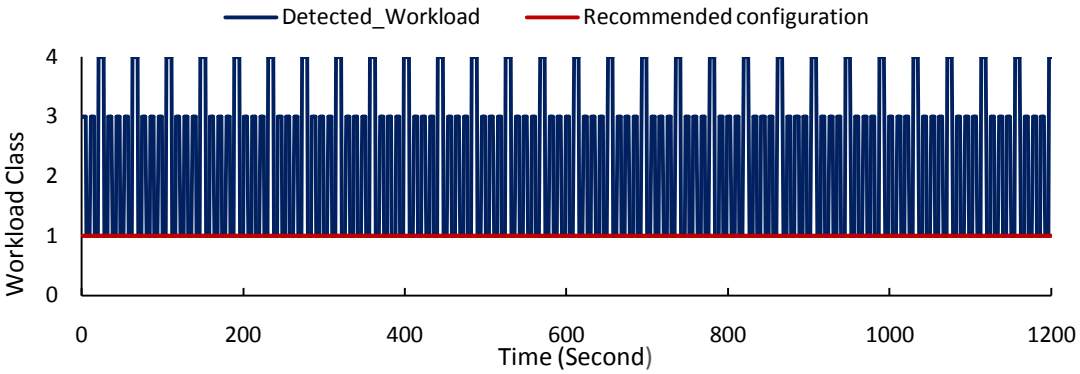

Fig. 5:detected transient workload in Transient status

4.2.2. Experiment 2: Cloud cost

The total cost includes the SLA violation penalty and cloud service usage for ASM and SSM approach that is depicted in Fig. 6 at first, the two approaches had equal costs but after a period of the time, the cost of SSM increased in comparison to the ASM. The results showed that the ASM had 52% reduction in total cost in comparison to the SSM.

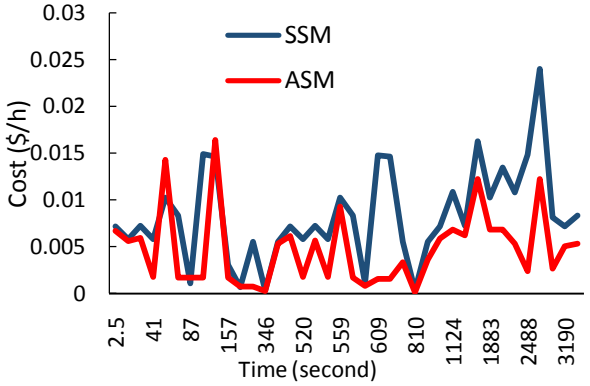

Fig. 6: total cloud cost

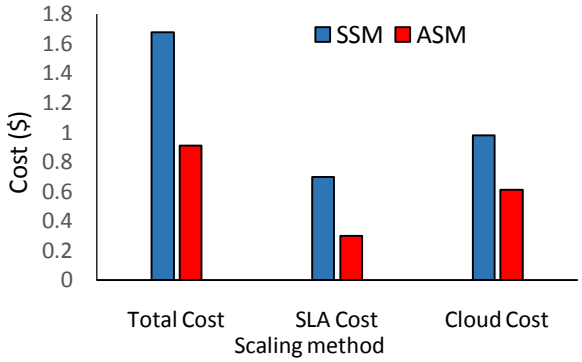

Fig. 7: cloud cost and SLA penalties

The cost of SLA violation and cloud service usage is shown in Fig. 7 the SSM approach couldn't satisfy the SAL requirements properly and total cost increased 62% in comparison to the ASM. The ASM could reduce the 59% of cost because of ignoring reconfiguration for transient workloads.

4.2.3. Experiment 3: Time

The duration of SLA violation is depicted in Fig. 8 the violation duration of ASM was only 12.2 seconds in an hour, while this time for SSM was 21.6 seconds in an hour. This percentage of SLA violation satisfies the service users as this is an important measure for service quality of could provider.

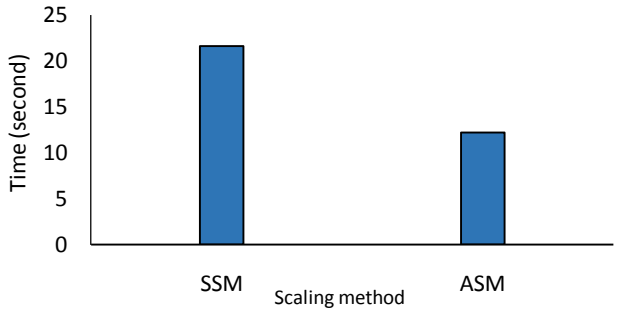

Fig. 8: virtualization manager APIs

4.2.4. Experiment 4: The number of software reconfigurations

In Fig. 9 the number of reconfigurations is shown for mixed and transient workloads. Due to more reconfigurations in SSM approach, the system goes to an unstable mode and the main reason of this problem is ignoring the transient workloads. However the ASM approach (as shown in Figure 6) was able to reduce nearly 50% of the reconfiguration times in comparison of the SSM method.

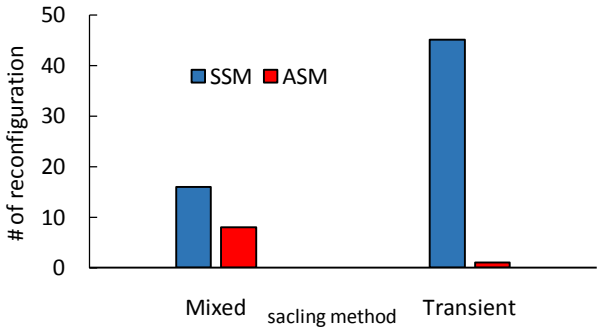

Fig. 9:number of reconfiguration

5. Conclusion

In this paper the problem of recognizing the transient workloads in the cloud service was elaborated. A self-management architecture was proposed to recognize the transient workloads and also performing the auto-scaling process by interfacing the OpenStack hypervisor. The Learning Automata (LA) method was used in the proposed architecture to learn the appropriate action during the load change. The scaling cost is also formulated in order to achieve a cost effective configuration for the applications in cloud. The proposed architecture was implemented using the Heat API's in the well-known OpenStack hypervisor. The experimental results showed that the proposed method had higher accuracy, lower configuration cost, minimized SLA penalty, reduced duration of the SLA violation and also brought the reduction in the reconfiguration times in comparison to the SSM method.

References

[1]C. Rong, S. T. Nguyen,M.G.Jaatun, "Beyond lightning: A survey on security challenges in cloud computing,"*Computers & Electrical Engineering*, vol.39, no.1, pp.47-54, 2013.

[2] J.Chen, G.Soundararajan, C.Amza, "Autonomic provisioning of backend databases in dynamic content web servers," *Autonomic Computing, ICAC '06. IEEE International Conference on*, pp.231-242, june.2006.

[3] C.Lin, J.Wu, J., Lin, L.C.Song, P.Liu, "Automatic Resource Scaling Based on Application Service Requirements, "*Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pp.941-942, June 2012

[4] T. C. Huang, S.Zeadally, "Flexible architecture for cluster evolution in cloud computing, "*Computers & Electrical Engineering*, vol.14, pp. 90-106, 2014.

[5] OpenStack Home Page. http://www.openstack.org/.

[6] S.Dutta, S.Gera, A.Verma, B.Viswanathan, “Smartscale: Automatic application scaling in enterprise clouds”, *Cloud Computing (CLOUD), IEEE 5th International Conference on*, pp.221-228, June.2012.

[7] J.M.Londoño-Peláez, C.A.Florez-Samur, “An Autonomic Auto-scaling Controller for Cloud Based Applications”,*International Journal of Advanced Computer Science & Applications*, 2013.

[8] J.Jian, J.Lu, G.Zhang, G.Long, “Optimal cloud resource auto-scaling for web applications“,Cluster, Cloud and Grid Computing (CCGrid), 13th IEEE/ACM International Symposium on, pp.58-65, May.2013.

[9] N.Roy, A.Dubey, A.Gokhale, “Efficient auto scaling in the cloud using predictive models for workload forecasting”,*Cloud Computing (CLOUD), IEEE International Conference on*, pp.500-507, July.2011.

[10] J.Yang, C.Liu, Y.Shang, B.Cheng, Z.Mao, C.Liu, J.Chen, “A cost-aware auto-scaling approach using the workload prediction in service clouds”, *Information Systems Frontiers*, pp.1-12, 2013.

[11] E.Casalicchio, L.Silvestri, “Autonomic Management of Cloud-Based Systems: The Service Provider Perspective”, In *Computer and Information Sciences III*, Springer London,pp. 39-47.

[12] J.Kupferman, J.Silverman, P.Jara, J.Browne, “Scaling into the cloud”, *CS270-advanced operating systems*, URL http://cs.ucsb.edu/~jkupferman/docs/ScalingIntoTheClouds.pdf

[13]T.Lorido-Botran, J.Miguel-Alonso, JA.Lozano, “Comparison of Auto-scaling Techniques for Cloud Environments”, In: *Alberto A. Del Barrio, G. B. (editor), Actas de las XXIV Jornadas de Paralelismo. Servicio de Publicaciones*

[14] F.RezaeianZadeh, A.Kumar Pandey,M.DavarpanahJazi. A.Kumar, "Capacity Planning Scientific Workflow on Cloud through Ant Colony Approach",*in Eighth International Conference on Computer communication networks (ICCN 2014)*, 2014, pp.19-25.

[15] O.Busherian, Z.Reisi, “iScale: An Intelligent Auto-Scaling Engine for Migrated Applications to the Cloud”, *ARPN Journal of Systems and Software*, vol.4, no.5, pp.116-122, August.2014.

[16]H**.** Beigy, M.R.Meybodi, “Learning automata based dynamic guard channel algorithms,” Computers& *Electrical Engineering*, vol.*37*, no.4, pp. 601-613, 2011.

[17] R.Oliveira,R.Vilaça, M.Matos, L. Beernaert, L.Beernaert, M.Matos, “Automatic Elasticity in OpenStack”, In *SDMCMM'12: Workshop on Secure and Dependable Middleware for Cloud Monitoring and Management, with ACM/IFIP/USENIX ACM International Middleware Conference*, 2012.

[18]https://wiki.openstack.org/wiki/Heat

[19] "Auto scaling with Heat and Ceilometer ". http://techs.enovance.com/5991/autoscaling-withheat- and-ceilometer.

[20] "Ceilometer + Heat = Alarming", http://www.slideshare.net/NicolasBarcet/ceilometer -heatequalsalarming- icehousesummit.

[21] H.Lee, G.vonLaszewski, F.Wang, G.C.Fox,“ Towards Understanding Cloud Usage through Resource Allocation Analysis on XSEDE”, *Community Grids Lab Publications*, march.2014.

[22] P. R.Gupta,S.Taneja,A.Datt, “Using Heat and Ceilometer for providing Autoscaling in OpenStack”, *International Journal of Information, Communication and Computing TechnologJagan Institute of Management Studies, New Delhi*, vol.2, pp.84-89, july.2014.

[23] "Heat/ AutoScaling", https://wiki.openstack.org/wiki/Heat/AutoScaling.

[24] "Heat Autoscaling API v1", http://docs.heatautoscale.apiary.io/.

[25] Amazon EC2, http://aws.amazon.com/ec2.

[26] A.Beitch, B.Liu, T.Yung, R.Griffith, A.Fox, D.A.Patterson, “Rain: A workload generation toolkit for cloud computing applications”, *Electrical Engineering and Computer Sciences University of California at Berkeley*, 2010.