A Review of Auto-Scaling Techniques for Web

Applications

|  |  |  |
| --- | --- | --- |
| Sara Al-Rasbi  200909464 | Nada Aboueata  200900288 | Rahma Ali  201001782 |

November 12, 2017

**Abstract**

The popularity of cloud computing resulted in the need to advance techniques to enhance its performance based on the load on its resources. Therefore, Cloud provider compete to deliver the users the best solutions to overcome performance issues by using auto-scaling techniques. Many techniques have been proposed, starting from basic methods that are threshold-based to more advanced techniques that employ real-time data and machine learning techniques that aim to auto-scale the resources without the intervention from the Cloud user.

# Introduction

With the increase use of web service applications, techniques to make cloud computing more effective and efficient while maintaining minimum costs to the service providers have started to garner significant interest. One of the ways that are implemented is by automatizing the different components of the cloud computing technology in order to address the dynamic demands of web applications. Applications using cloud computing can be categorized into three main categories [1] namely infrastructure-as-a-Services (Iaas), Platform-as-a-Service (PaaS) and Software-as-a-Service (SaaS). In IaaS, the cloud provider provisions all infrastructure components needed for the server deployment. These components could be networks resources such as the processing, storage, bandwidth requirements among others. One of the most common examples is Amazon EC2.

Elasticity is one of the main key characteristics of cloud computing. Due to its elastic nature, users can acquire and release resources as needed. However, if the number of resources assigned is not adequately set when there is a dynamic change in the number of the resources, this can lead to over-provisioning or under-provisioning of resources. Even though users can allocate resources dynamically according to the current demands, it is nevertheless still challenging to decide the right number of resources. Therefore, it’s necessary to develop a sophisticated technique for resource allocation which should consider the application demand. These techniques are known as auto-scaling techniques in the literature.

Auto-scaling techniques are broadly divided into two categories - reactive and proactive. The reactive techniques are based on a set of predefined rules. By using these rules, the system reacts to workload changes only when those changes have been detected. The main issue with the reactive approach is the time it takes to activate the allocated resources and make it ready for useOn the other hand, proactive based techniques attempt to predict the future workload and prepare the required number of resources ahead of time.

As mentioned earlier, resource allocation is usually performed by an auto-scaler. Resource scaling can either be done horizontally or vertically. Horizontal scaling -also known as scale out or scale in-refers to adding or removing server replicas that are running on VMs. On the other hand, vertical scaling refers to changing the number of resources by increasing or decreasing the number of allocated resources such as CPU, Memory and others to an already running virtual machine.

Any auto-scaler must guarantee the correct functionality of the application, by providing an acceptable level of Quality of Service (QoS) measure; such as fast response time, high throughput, and high availability to the end users. Without such assurances, the Service level agreement (SLA) between the customer and the service providers will be lost. Typically, maintaining the SLA depends on the degree of satisfying the QoS properties. In any case, the auto-scaler should be able to address the trade-off between minimizing the cost of resources allocation and maintaining the SLA by assuring the highest possible degree of QoS properties satisfaction.

Moreover, any auto-scaler system should implement the MAPE loop autonomously. MAPE consists of four main phases: Monitoring, Analysis, Planning, and Execution. In the context of auto-scaling, a monitoring system will be responsible for gathering historical data about the system; this includes performance metrics such as hardware counters and application log information. Then, the Analyzer model analysis the retrieved data to predict the future values; later on, the planning model will plan a proper resources allocation decision. Finally, the provider executes the plan of auto-scaler accordingly [1].

In this paper, we review and critic different auto-scaling techniques that are proposed by researchers of this field. In our review, we adapt the classification done by [1], as the authors propose to classify the methods of solving the scalability problem on a a different type of classification rather than using the well known broad categories - reactive and proactive. The rest of the paper is organized as follows: section 2 includes the review of the literature on auto-scaling techniques given the classification categories suggested by [1]. We start with Threshold-based rules in section2.1, discuss Control Theory in section 2.2, and Queuing Theory in section

2.3. Then, we discuss papers that implement Reinforcement Learning techniques in section 2.4, and Time Series Analysis in section 2.5. Finally, we compare between the auto-scaling categories and discuss their strength and weakness in section 3. Then we conclude this review in section 4.

# Classification of Scalability Techniques

## Threshold-based Rules

Threshold-based rules are techniques that were very popular with cloud providers. they are very simple methods that define policies and rules that trigger the scaling process. However, these approach depend on the thresholds defined as they need to be set accurately to avoid wrong scaling decision furthermore the crafting of rules require a good understanding of the workload and its trends. As in any rule, rules have two parts: a condition and a consequence or an action. In scalability problems, the condition usually include performance metrics such as CPU load or request rate, etc. typically each metric could have an upper threshold and a lower one. If any of the thresholds is reached, it means a scaling process might be needed [1].

In the works of [2], a system called Integrated and Autonomic Cloud Resources Scaler (IACRS) a cloud resource allocation system is proposed. It uses threshold-based methods, and it also takes into consideration performance metrics from different domains (compute, storage, and network) to make the scaling decision, as the author argues that the techniques at that time did not consider the metrics in a combined matter, but instead they were isolated during computations.. The inputs for the system are: the resource it needs to monitor and auto scale when needed, the performance metrics to monitor, a reference to a resource if the metric is binary. Group Id of the group the resource belongs to. A group can be defined using different criteria to help in the scaling process where the scaling is applied to all resources of a group; for instance, grouping by resource type. the last inputs the system needs are the threshold policies and the auto-scaling policies. The proposed solution added two new thresholds parameters that are set between the initially defined threshold; one is defined lower than the upper threshold while the other is defined above the lower threshold. They also make use of the duration parameter to check the metrics and how persistence they are around the threshold. When the parameter metric exceeds the upper threshold for the first time, the observation period starts. If the value of the metric fluctuates between the upper threshold and the second upper threshold right below it until the end of the observation, then trigger scaling. Similarly, if the parameter metrics fall below the 1st lower threshold and it keeps oscillating between the two lower thresholds, a scaling down is triggered. The benefit of the grouping will be seen in this step as the scaling will be done to all resources under the same group.

Threshold-based rules, were indeed popular that they ever implemented by Amazon. However, there shortcomings were becoming more evident as the Cloud became more advanced. The Cloud providers needed a better solution for scaling that adapts to the changes on workload, and utilizes the resources accordingly. Thresholds defined are fixed, and the rules are crafted at a specific time thus they might be outdated after long usage. therefore, this technique alone is not effective to be used in a Cloud computing environment. However, some researchers use similar aspects like defining rules or threshold while incorporating different techniques to overcome the weaknesses of Threshold-based methods.

## Control Theory

In this section, we discuss auto-scaling techniques that adapt control theory to achieve its objectives. In control theory, a controller is defined to monitor a variable, and according to the value of the variable, the scaling method will decide to either add or remove resources. The controller could be open-loop, feedback, or feed-forward. Each type is suitable for different scenarios. An open-loop controller uses the current state of the system and its model to determine the input for a method - the number of resources e.g.number of VMs- to achieve the optimal or the targeted system. Whereas, feedback controllers observe the system outputs -in terms of performance and load balancing- if they are close to the desired goal or not, if not the controller will keep changing the inputs to achieve the desired system output. Feedforward controllers are for predictive techniques. They try to predict the issues and react before they occur. Usually, feedback and feed-forward controllers are combined to reduce prediction failure.

In the work of [3]. The authors make use of threshold-based techniques in combination with control theory to propose an adaptive method. Thus, it makes use of their performance in accuracy and precision without having to set fixed thresholds. They proposed two different methods, one for scaling up and the other for scaling down. Both methods monitor the virtual resources. The main idea of their approach is not to scale directly after a threshold is met, but to observe the variable for a predefined interval. The purpose of this observation period is to decide if the scaling is necessary; thus, avoiding any unneeded scaling which eventually means better utilization of resources and reduction of costs. As shown in the Figure 1, data is being regularly collected from sources like CPU or memory; then they are used to derive a decision to scale or not by the data analysis component. In the algorithm, they have two parameters *α* and *β*. The parameter *α* is used to increase or decrease the initial threshold T depending if it is scaling up or down. Whereas, *β* determines the monitoring period when the threshold is exceeded.

In scaling up, after analyzing the collected data. If the threshold is met the controller will monitor the system state for *β* seconds, if the value of the collected data -CPU or memory utilization- is still above the threshold, the value of the threshold is increased by *α* percent. Otherwise, if during the observation interval the value of the performance metric falls below the threshold, the threshold is left unchanged. This scenario continues, while the overall threshold does not exceed the full utilization possible, once the threshold reaches the highest and is equal to the full utilization value, the scaling up is triggered. In this algorithm the selection of the initial threshold and *α* is critical, they need to be selected carefully so that the system does not reach the 100% utilization threshold (full utilization) in the first run of the algorithm.

In case of scaling down, the authors define a parameter called *Tmin* that indicates the minimum tolerance level that the threshold could reach. Similar to the method of scaling up, once the utilization of the VM resources reaches the threshold the monitoring phase starts. In this phase, the algorithm will observe the situation for *β* seconds, if by the end of this period the utilization level is below *Tmin* the scaling down is triggered. If not, and the utilization level is below the threshold, the threshold is decreased by *α* percent temporarily. Then the method continues the observation for another *β* second. If by the end of this period the utilization level of the resources is less than the threshold but greater than *Tmin*, then the changes are acknowledged, and the threshold is decreased by *α* percent permanently and used as the new threshold for the next iterations. In order to decrease the false positives, where unneeded

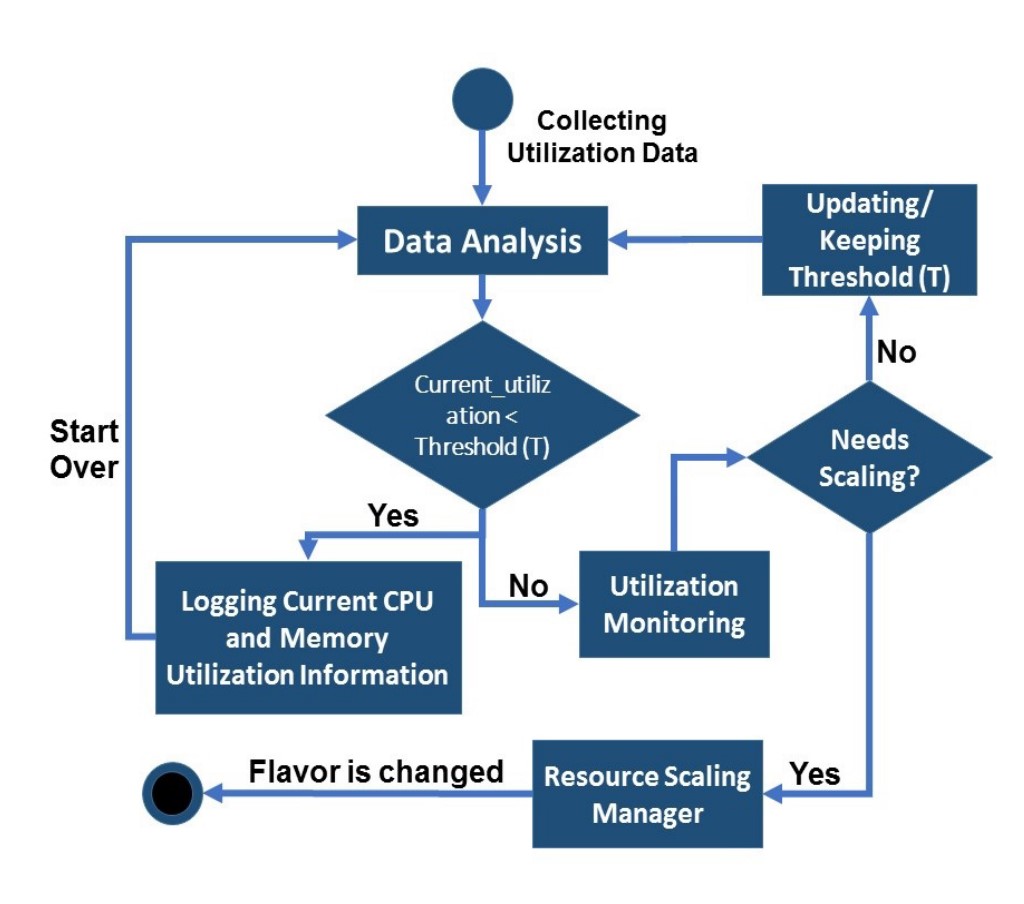


Figure 1 The Decision making in the proposed method.

scaling is triggered. They use a “Ripeness Function" (RF), for situations wherein the second analysis period the CPU or memory utilization might exceed the threshold. As the authors state that this event could be the result of a temporary burst that might be stabilized quickly, thus it should be handled with care, so it does not harm the ongoing process of observation. Therefore, using the RF they force an extra observation for another *β* seconds. In this period, the value of the threshold will be updated and committed if at the observed value of the resources utilization level at the end is the same as the observed value at the start of the monitoring period.

To evaluate their algorithm, the authors installed Wordpress on a VM with two CPU cores and 2GB RAM. As a test, they simulate the performance of CPU and memory utilization as a varying number of users connect to the VMs. They evaluated their algorithm against a simple threshold-based algorithm. In the results of testing CPU utilization, the threshold based method increases the number of cores the moment it detects the need for scaling, while their algorithm did not find the need to do any scaling. As for memory utilization, in the downscaling test, the threshold once again signaled a scale down as soon as the test started as the number of users was low; however, the proposed algorithm did not scale down as it saw high utilization during the observation period.

As mentioned this technique improved the fixed threshold in threshold-based methods, as it makes use of real-time data to make the necessary changes on the threshold; thus, avoiding any unnecessary scaling the moment a threshold is reached. However, the algorithm does not indicate how it decides how many new resources, it just mentions that scaling is happening.

Also, the performance of this method relays on the initial threshold, and *α* if they were poorly selected this algorithm will fail to scale correctly.

## Queuing Theory

The arrival of request to web applications, and its associated information like waiting time, queue length, distribution of the request arrivals can be benefited from to scale web applications. Queuing theory is a study of queues in mathematics. In a typical queue, request come to the system at an average arrival time *λ*, each request is enqueued until its serviced. A single queue can be serviced by a server of different servers[1]. [4], he employed queuing theory to achieve auto-scaling in web applications. The motivation behind his work was to address the inconstant usage of reserved resources. They argue that the number of accesses to a web application is not stable and its oscillate over time; this could lead to a phenomenon referred to as peakvalley phenomenon. This phenomenon states that *"The amount of reserved resources is often proportional to the peak needed of physical resources, while most of the time the amount of required resources is far below the peak load and thus physical servers will be idle for most of the time."* Therefore, they proposed a queuing model M/M/C, which represents a multi-service window and infinite source system. In queuing theory, the notation used to describe the system is written as follows: A/B/C/X/Y/Z:

A: Distribution of request arrival time

B: Distribution of service time

C: Number of parallel servers available

X: Capacity of the server - how many requests a server can handle, including the ones already in service

Y: Calling population - type of the population the requests are coming from its referred to as open if the source is infinite and closed if it is finite.

Z: service disciplines-defines the priority order the request should be served ( e.g., First In First Out)

The algorithm can be summarized as a model that can predict the arrival time of requests and calculate the minimum amount of resources to reach the level of resources demand. Then use dynamic programming or heuristic algorithms to design VMs autoscaling strategies either horizontally or vertically. The queuing model the authors adapt shown in Figure 2 is noted as M/M/C, Where the request arrival time (M) follows a Poisson distribution. The service time (M) follows an exponential distribution. While C denotes the number of servers. In practice, such system is called a system with a multi-service of a queuing model M/M/C. The system also includes a load balancer that (LB) that serves as the queue used for the requests until a server is ready to serve the request. Using the mathematics of query theory, the authors define equations get the performance metrics of the system in a time *t*; such as idle probability of the system, average number of request in a queue, average number of request handled by the system, expected wait time in the queue and the expected wait time in the system. The problem the authors attempt to solve is once given the state of the system in a time interval how to decrease the demand for resources of the system and the waiting time in the queue. The resource allocation algorithm the authors propose computes the optimal number resources (*Vt*+1) needed in a time interval(*t*+1) with respect to the state of the system. As an input for the algorithm, it needs the traffic intensity, idle probability of the system, average waiting time in queue (*Wq*) and the system(*Ws*) thus the objective function will be:

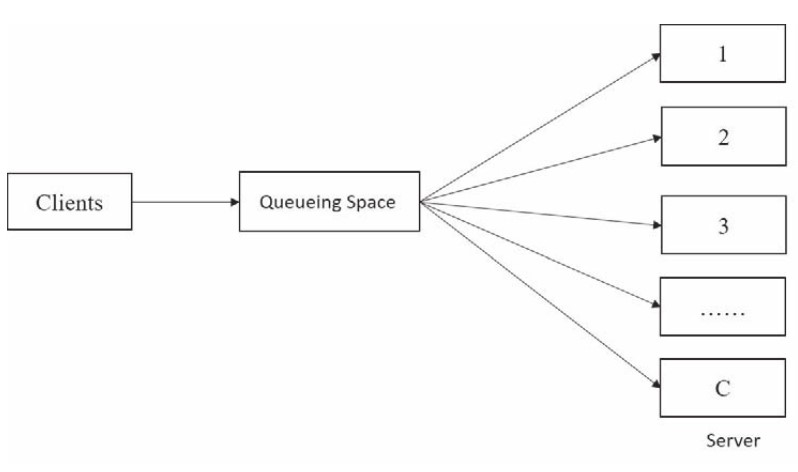


Figure 2 The Queuing Model M/M/C.

*F*(*V t* +1) = *Vt* +1+ *α.Ws*(*Vt* +1)

That is a discrete function with a U-shaped trend. As for the auto-scaling of VMs, an algorithm determines if the system should be scaled up or down, and how much to scale. Using the resource allocation after getting the value of optimal resources *Vt*+1 if its greater than the current resources *V* then the system needs to be scaled up; otherwise, if *Vt*+1 *< V t* the system needs to scale down.

In case of scaling up a heuristic function is used to determine which VM needs scaling and how much to scale. Basically, in the algorithm, if A is the total of resources remaining for each VM in a host and B is the max resources the VM support, if there are k VMs in the same host that resources can be increased, A is changed to the average of the remaining resources. Thus, the new A is the average of the previous A and the rest is assigned to the VM while respecting B. As for scaling down, they only consider horizontal scaling as vertical scaling might lead to failures. As a result, the scaling down becomes a 0-1 knapsack problem that’s solved by dynamic programming. Where the problem to solve is to select which VMs to close while keeping the resources greater than *V* − *Vt*+1.

When the proposed solution was tested on OpenStack, They used multiple threads to represent the VMs and defined a workload to test how the scaling up algorithm work. Initially, 4 VMs were created with the resources 4,3,2 and 2 respectively. While testing an average number of request of 500, the VMs were heavily overloaded. Then, the resource allocation algorithm and the scaling up changed the resources in each VM to 5,5,4 and 2. After the change the load on the virtual machines reduced. To test the scaling-down algorithm, the average number of requests was set to 300 while the resources on the 4 VMs were 4,4,6 and 7. The load on the 4 VMs was very low thus some VMs needed to be shut down as the system was not utilized correctly. Thus the scaling-down algorithm and the resource algorithm will choose to close or terminate the first and the fourth virtual machines, leading to a better utilization of the resources and larger load on the remaining VMs. the implementation of queuing theory to solve auto-scaling techniques lead to great results. It is understandable why this theory acquired such results, as the workload on the serves is viewed as a queue and make use of its simplicity to craft solution to the auto-scalability problem. However, a queuing theory solution can be an applicable. A queue that based on exponential distribution must have a variant coefficient of one, otherwise, the queuing theory wont be applicable to use.

## Reinforcement Learning

Reinforcement learning (RL) is another technique used in auto scaling. RL is closest in concept to the control auto-scaling technique and only differs in the fact that unlike in control theory, the automation of the scaling task is not dependent on any pre-existing knowledge or any performance model of the application [1]. It is for this reason, auto-scaling using this technique has started gaining popularity in the literature. In RL, scaling decisions are made and improved on while the application runs. The auto scaler component in RL is known as the agent. When a state change occurs in the application, the agent can choose to perform an action - which could either be to scale horizontally, to scale vertically, or to perform no action at all. These actions are selected based on the current state of the application and its workload and response rate. In RL, the action chosen for a given state is then rewarded according to the nature of the succeeding states. That is, if the application is in better situation, in terms of performance, after an action is performed, the committed action is then considered optimal for the time being and the action is positively rewarded and vice versa. In essence, it is this rewarding system that drives the agent to select the appropriate actions for given application states. Intuitively, the agent learns and assigns actions to different states in such a way that maximizes collected rewards. Because an action could result in both a positive or negative reward, learning what action to perform and thus auto scaling on a whole is done using a trial and error approach.

One way the learning process is implemented in the literature is by first mapping all system states to all possible actions. The aim of the agent, in this scenario, should be to find a policy that maps every state in the application to its respective best action [1]. Each state-action pair is assigned a value called the q-value. The better the nature of the state achieved, the higher the reward obtained for an action performed, the higher the q-value assigned to the state-action pair. In this type of learning process, all the q-values are initially all set to the same value. As learning progresses, these values are updated. In the end, a greedy approach is followed in selecting the appropriate actions to move from one sate to another. The appropriate actions are those actions that belong to pairs with higher q-values which are ultimately selected the next time the same state is reached. While using RL techniques for auto-scaling seems appealing in that it can easily be plugged and used in any application on the go, it nevertheless has its drawbacks. One of its major drawbacks is that it is a time-consuming process due to the time it takes for the learning process to learn the most appropriate action for each state. Because it is usually done on the fly using a trial and error method, the learning process may become impractically long in some cases. In order to understand how RL techniques are applied in the literature, we reviewed the implementation of RL techniques in a single layered application and in multi-layered web applications.

One such research where a technique using RL as the auto-scaler was proposed was in [5]. Arabnejad et al in their paper extend a previous work that uses fuzzy logic to auto scale and combine it with two RL techniques - Q-learning and SARSA - to present two different auto scaling models. By using RL techniques to the output of a fuzzy logic auto-scaling controller they intend to make the process of allocating resources dynamic. The motivation for their work is to compare these two proposed models in terms of meeting QoS requirements in a cost-effective manner. They implemented their proposed models on OpenStack and evaluated them used multiple workload patterns.

The controller they used in the prediction utilizes Fuzzy logic. Fuzzy logic is an approach in computing that looks into the degree membership of an object to multiple sets rather than assigning objects to a fixed number of sets. This idea was inspired from human nature of making decisions. Unlike in digital systems, decision making in humans is more inclined to being based on not crisp boundaries. As a result, the defining boundaries or the labels of what each set includes is not clearly set and is fuzzy or unclear in nature. As a result, in a fuzzy inference system, since the boundaries of sets are not crisp, multiple states can be grouped together to get a fewer number of states. Which is one of the strengths of using fuzzy systems since the space required to store the earlier mentioned Q-values is significantly reduced.

While fuzzy logic plays the role of the controller which predicts the resources required, RL techniques are used in the auto scaling technique. Q-Learning and SARSA are the RL techniques they selected to model their proposed solutions. These techniques differ in their approach of how the next state is selected namely on-policy or off-policy. While Q-function compares the current state to the best possible next states making it an off-policy technique, SARSA on the other hand just compares the current one with the actual next obtained from pre-defined policies making it thus an on-policy technique. While both the techniques have their strengths and weaknesses, in the case of web applications, SARSA seems to be a better fit since it follows policies, thus in theory would already be proven to be more optimal and result in faster learning process.

Their proposed model is composed of three main components – the monitoring component, the fuzzy controller and the learning module. The role of the monitoring component is to continuously monitor the different characteristics of the application such as workload, response time and number of virtual machines. These characteristics are later fed into the fuzzy controller and learning component. The monitoring component also verifies that the system goals, such as minimizing costs and response time and respecting the SLA, are met and resources are effectively allocated. Meeting system goals will accordingly be reflected on the rewards function. The role of the fuzzy logic controller is to decide the appropriate scaling action to be done after considering the monitored input data and the set of pre-defined rules. Scaling is done by incrementing or decrementing the number of virtual machines. Since the focus on this paper is on the auto scaling techniques presented in different works, we will focus on how the process of auto scaling is performed in their proposed system.

In their proposed solution, a state, s, is represented as

*s* = (*w,rt,vm*)

where w,*rt* and *vm* stand for workload, response time and number of virtual machines necessary respectively. How auto scaling is performed in this schema is discussed as below –

1. The q-values need to first be initialized to 0. Each of these values represent a mapping or a rule that is assigned to state action pair and is updated during the learning process.
2. Next step is the exploration/exploitation stage. In this an action is selected. An action with the best reward is selected most of the time, with random actions selected with a significantly lesser probability. This random selection is made in order to let the learner explore non-visited actions. As the number of actions selected increase, the probability of selecting random actions decrease until the exploration phase is stopped.
3. Then the control action triggered by the fuzzy controller is examined.
4. Next the Q-function value is calculated from the so far obtained current q-values. Because a fuzzy system is used, each action could have multiple rules unlike in more traditional methods where every action state pair has only rule or policy.
5. The controller then gets the current count of vms and response time of the state. Using these two the reward is then calculated. Two measures are considered in this calculation which are the number of resources obtained, which regulates the cost, and the SLO violations committed.
6. In the case of Fuzzy SARSA Learning (FSL), the q-value of the new state is then calculated by considering the actions available to be chosen from and select the maximum q-value applicable to the new state obtained from the multiple fuzzy inferenced policies. However, in the case of Fuzzy Q-learning technique, the next selected state is selected in such a way as to get the largest possible reward.
7. Once all these steps are done, the q values are then updated.

All of the above steps are repeated until convergence is achieved.

In order to evaluate their work, they simulated different types of workload patterns. There are three types of representative workload patterns namely the Predictable Bursting pattern, the Variations pattern and the on-and-off pattern. Predictable Bursting pattern is a workload pattern where in the workload undergoes periodic spikes during certain known times. This type of workload is normally seen on servers that offer services with seasonal trends. Variations pattern on the other hand is another type of workload pattern where the spike of the user requests is not predictable and happens in a random fashion. This is usually seen in news applications and event registration applications. Lastly, on-and-off pattern is the most stable of all of the workload patterns and reflects applications that have a well known and fixed usage period, such as test environments.The authors have tested both their proposed models with all of these workload patterns.

In the case of predictable burst workload patterns, FSL performed significantly better than FQL because given that FSL is an on-policy learning model, it was faster at converging than FQL. That means, although FQL selects the best policy, FSL is better at reaching a balanced exploration/ exploitation phase since it is an on-policy technique. As a result, in this scenario, SARSA has a better tendency to get a more accurate number of VMs launched faster. However, the opposite is the case for the Variations workload pattern. FQL performs significantly better than the FSL approach here. Interestingly, it is due to the same reason of FSL being an onpolicy model and thus making the learning and scaling process faster that makes it perform badly in scenarios where there are quick variations in workload resulting in random fluctuations and non-periodic behavior. Lastly, in the case of the ON and OFF workload patterns, both of the models performed similarly.

In order to evaluate their proposed solutions, they evaluated by comparing them for cost effectiveness, they tested their proposed models to determine the number of VMs used for the different workload patterns. While, FQL and FSL were both able to conduct scaling and allocate suitable numbers of VMs for different types of workload patterns, FQL was better at performing resource allocation of VMs. They estimated the average maximum number of VMs used by FQL at 18.3% while that allocated by FSL at 22.6% for the same workloads. This also shows that both their models were able to meet the QoS and SLA requirements by using a significantly smaller number of resources.

On the other hand, Iqbal et. Al in their paper [6] attempted to address auto scaling using RL for multi-layer web application. Their proposed solution is a complete auto scaler and has both workload pattern prediction and resource provisioning policy learning. Their proposed solution could be used by cloud providers that need to build systems that can automatically manage their multi-tier web applications while respecting the assigned SLOs. All of which should be done without any prior knowledge of the applications’ resource utilization or workload patterns. Since for this paper, we are only focusing on policy learning techniques and how auto scaling is done, we will in the next section, only explain the policy learning technique proposed by them.

Iqbal et. al. in their proposed solution proposed an online unsupervised method for policy learning for adaptive resource allocation of multi-tier Web applications. They set their learning agent to use a simplistic method to finding the policy in such a way to maximize an objective function which rewards satisfying a response time adherence, SLA contracts with minimal resources utilization as possible. Before discussing the algorithms, we will first look at their definitions of state, action and policy. They defined the system state at time t as

*st* = (*U,P*(*C*)*,λ,p*)

where U is the configuration of the Web application’s tiers. Since multi-tier applications are considered, U is defined by a vector U = (*u*1, . . . , *un*), where *ui* indicates the number of machines allocated to tier i in the application. P(C) is the current workload pattern of the application, lambda, the arrival rate of the requests and p is the 95th percentile of the service time for the application. These are taken for a known time period. Their auto scaler was designed to only do vertical scaling but at different tiers. So according to need, the agent could scale up the Web tier (*aw*), scale up the database tier (*ad*), scale up both tiers (*ab*), or do no scaling (*aφ*). Thus, the set of possible actions that the agent could perform can be shown as

*A* = *aw,ad,ab,aφ*

As in most reinforcement learning techniques, they consider the policy to be a value function that maximizes the reward. This function does so by predicting the value of each possible action and selecting the one with the highest predicted reward. With respect to the algorithms, their implemented their approach in 2 algorithms, one for the exploration, which is the online learning phase and other for the exploitation phase which is the decision-making phase. The process of scaling of resources is triggered when an SLA violation is encountered. For the exploration phase, starting with no knowledge, the learning agent periodically monitors for SLA violations. If the agent detects any violation, it selects the action with the highest reward by attempting all possible actions. Exploring of all these actions is done using a simple exhaustive exploration algorithm. Each of the states visited during this phase are logged and later used to build a neural network regression model. This model will then be used to predict the service time required for by the auto scaler during the exploitation phase for a given workload and tier configuration. During the exploitation phase, when an SLA violation is perceived, the value of each action is calculated by first predicting the service time that the action would take by using the regression model and then by calculating the reward. Once that is complete, the action with the maximum reward is selected. They maintain a pool of already booted VMs to quickly add to a specific tier of application when the need for them arises.

In order to evaluate their work, they used they implemented their policy learning module on a two-tier web application installed on Amazon Elastic Compute Cloud (EC2) and RUBiS, a benchmark Web application for auctions, as a sample web application. They tested their models on two experiments – light workload pattern and heavy workload pattern. For the former case, they modeled each user session to have 32 user requests with the use of less number of resources required to complete their tasks. In the case of the heavy workload experiment, they modeled each user session to have 10 user requests and requiring intensive resources that require a long time to process. For both, they started with 8 user sessions and increased it every 5 minutes while repeating the same workload generation steps.

They analyzed and compared the performance of their policy learning methods – exploration and exploitation with the industry standard rule based methods CPU reactive and response reactive. The performance metrics they looked at were the total number of allocated CPU hours and the percentage of requests violating the SLA.

Their proposed solutions allocated fewer CPU hours in the policy learning stage for both experiments, with 11.67, 11.12 for exploration and exploitation phase respectively while 24.27 and 15.57 for CPU reactive and response reactive respectively. However, in terms of requests violating the SLA, they had a higher percentage with 1.32 and 0.58 for exploration and exploitation phases respectively and 0.68 and 0.89 for CPU reactive and response reactive respectively. Their argument is that, given that in both experiments, since the workload is always increasing, the other comparative methods perform overprovisioning of resources so as to avoid increasing the number of requests that violate the SLA, thus making them better at minimizing the number of SLA violations. However, this is not a favorable solution since it increases costs for the users of the cloud service. Therefore, there is a trade-off between user cost and SLA violations.

One of the strengths of their proposed methods is that since the input function considers both the raw arrival rate and the workload pattern, the value-function can have different values for the same state-action pair depending on the arrival rate of requests and how intensive the workload pattern is. It is because of this flexibility, that the learner is better able to provision resources. The basic guideline of the reward function is to encourage the learning agent when it respects the SLO while utilizing minimum number of resources and vice versa.

Another strength in this paper is their scaling strategy in terms of what tier in a multi-tier application is scaled. Interestingly they prioritize scaling out at the web tier over database tier when both actions result in achieving the same response time. Their argument for this is scaling the database tier introduces overhead of load balance at the web tier and data synchronization in the database tier which can be avoided by scaling the web tier.

While this paper offers compelling arguments, it also has weaknesses. One clear one is that they proposed their solution with an assumption that their system would always have sufficient bandwidth and that enough time would be given to the auto-scaler to collect access logs from the application, in order to learn and implement a policy before workload changes occur.

## Time Series Analysis

In the context of elastic applications, allocating the right amount of resource becomes a necessity nowadays given the sudden workload burst where the traditional scalers cannot react fast enough to this type of workload. Therefore, many researches in the literature have been addressed this issue using Time series analysis techniques. Generally, the auto scaling problem can be divided into two main steps: prediction step that predict future values and decision-making step for resources allocation. Time series analysis can be only applied to the first step, which is the prediction phase. Based on this prediction, the second step allocate resources accordingly. In the literature some of the research studies focus only on time series workload prediction techniques, while others focus on both, workload prediction and resources allocation. In this review, we discuss both types of studies, as the accuracy of resource allocation is influenced by how accurate the workload prediction algorithms.

Generally, time series techniques identify repeating patterns or predict future values by analyzing historical dataset. In the context of auto-scaling, performance metrics (such as CPU utilization, Memory usage, requests arrival rate, and others) are periodically sampled at fixed time intervals. The result will be a sequence of the last q observations sampled at a predefined window size w, that will be denoted as input window or history window. Then, it identifies the pattern (if any) followed by the time series and extrapolate it to predict future values. The predicted values of performance metrics are then used in the planning phase, to make the final decision of allocating resources [1].

Time series analysis have been applied mainly to predict workload or resources utilization. It can be classified into two main groups: techniques that focus on direct prediction of future values, while others extract pattern followed by the time series (if present) and then extrapolate it to predict future values. Some of the techniques that classified into the first group are: Averaging methods, Auto-regression, Auto-regression Moving Average, and different machine learning approaches [1]. The averaging method is the simplest way to predict future values. It calculates the future values either by treating all last q observations equally and compute the arithmetic mean (i.e. Moving Average), or by assigning different weights for each observation, in which more weight is given to the most recent observation and less weight to the older observation (i.e. Weighted moving Average) [1]. On the other hand, the techniques that classified into the second group are: pattern matching, signal processing and auto-correlation.

Most of the proposed approaches in the literature use either horizontal or vertical scaling to allocate resources, while in [7], Song Wu et. al proposed a hybridScaler algorithm for resource allocation which combines between horizontal and vertical scaling. This work considers both scaling approaches as each has its strengths and weakness. Reactive horizontal scaling techniques cause an overhead due to the time needed to boot the newly added VMs. Alternatively, proactive horizontal scaling can be used to overcome this issue. The proactive horizontal approach predicts the workload ahead of time and made the decision of resources allocation before causing any SLA violations. Despite its success in eliminating the overhead, but still it causes extra cost and significant overhead specifically with the short-term bursting workload. In reality, short-term bursting workload is hard to predict, and it requires to add and remove VM very frequently which cause significant overhead and extra cost. Therefore, proactive horizontal scaling is not suitable for frequent short-term allocation. The paper addressed this problem by using a lightweight reactive vertical scaling approach with the short-term workload, in which it adds/removes VCPUs and RAM instantly.

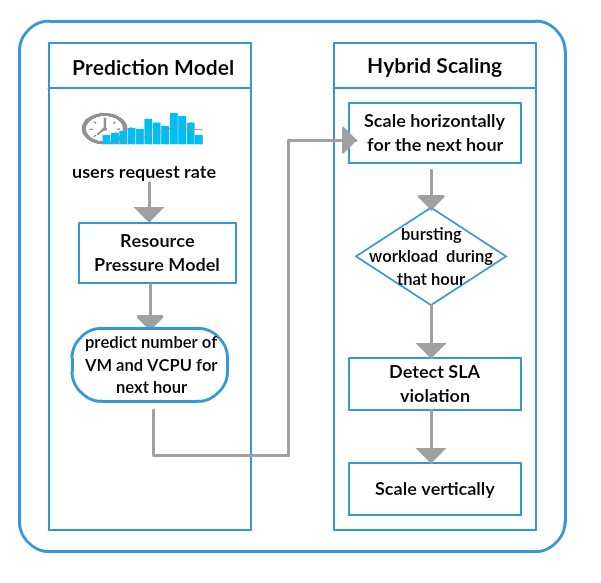
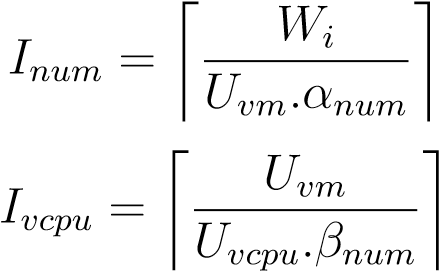


Figure 3 The HybridScaler architecture

As shown in Figure 3, they proposed a resource-pressure model and hybrid auto scaling algorithm. The resource-pressure model takes the request rate average (i.e. workload) as an input and predicts the appropriate number of resources, VM and VCPU, for the next hour to allocate for multi-tier application. This is done by predicting the workload pressure per each tier (*Wi*) and per each VM in that tier (*Uvm*). They used the Sparse Periodic Auto-Regression [8] which is a simple time series analysis method, to predict the workload. Thereafter, they use below equations to calculate the required number of

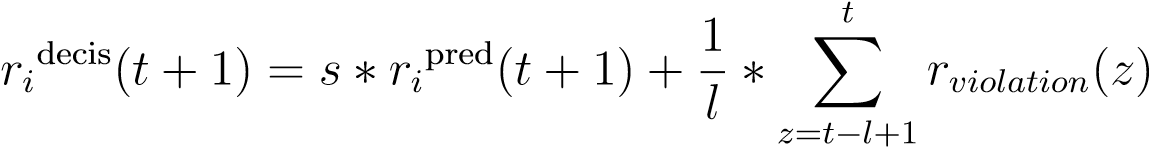
VCPU (*Ivcpu*) and VMs (*Inum*) either to scale horizontally or vertically.



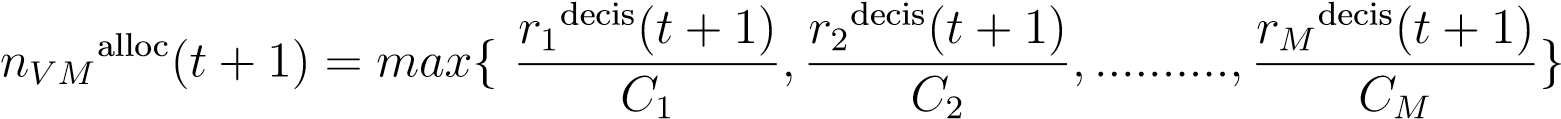
Where *Uvm* , *Uvcpu* shows the predicted workload pressure per each VM and VCPU respectively. And *Wi* refer to the workload pressure per each tier i. Thereafter, the hybrid auto scaling algorithm uses the computed number of VM and applies a horizontal scaling for the next hour. If a short-term bursting workload occurs during that hour and it exceeds the pressure of current resource pool, then the algorithm will detect SLA violations and the vertical scaling action is taken to react to the burst workload.

As for evaluation, they used an online benchmark system, RUBiS. The workload was generated by using the ClarkNet trace which has two weeks access logs. They used only the request rates for the first week and then average all rates per hour to train the proposed prediction model. As for comparisons, they implemented three baseline algorithms: two threshold-based horizontal scaling algorithm (one of them is designed to save resources and the other to provide high performance), while the third baseline is an hourly workload prediction-based horizontal scaling algorithm. They compare the proposed algorithm with the baseline algorithms from two major aspects, effectiveness and efficiency. The efficiency is measured by the response time and SLA violation, while the effectiveness is measured by the resource allocation amount and CPU utilization. The results shows that the proposed algorithm outperformed the baseline by reducing 16-39% in average response time and 34-50% in SLA violations rate. In addition, it manages to keep the CPU utilization between 60% to 70%, to avoid any resource wastage.

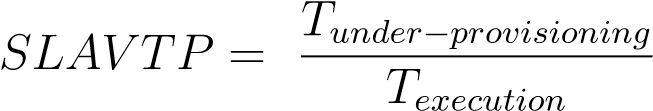
Similarly, to the previous work [7], Dang Tran1 et. al [9] proposed a proactive auto scaling solution that uses the SLA violation mechanism to adapt the decision of resources allocation. This work, develop a comprehensive auto scaling schema which includes a prediction model and scaling decision model that uses the SLA violation estimation to make the final decision of resource allocation. As opposed to the previous work, they proposed prediction model which exploits resources usage such as CPU and memory simultaneously. For example, the CPU usage is predicted by considering the relation between all historical performance metrics such as CPU and memory usage. So, by that way they avoid missing any relationship among the performance metrics in the prediction phase. The proposed prediction phase consists of three main sub-components: preprocessor, learning algorithm and forecasting model. The preprocessor component use fuzzy approach to transform the monitored performance metrics into a fuzzy time series, which are the input of the learning component. They use the Neural networks algorithm to build the training model. Thereafter, the forecasting model uses the trained model to predict the future values of performance metrics. The output of the forecasting model will defuzzied into real values, which are the final output of the prediction model. As shown in Figure 4, then the predicted performance metrics are then passed to the scaling decision model to make the final decision of resource allocation. To perform the scaling action, the amount number of each resource type i (*ri*decis(*t*+1)), such as number of CPU and Memory resources, is calculated using below equation:



Calculating the amount number of each resource type is based on two terms. The first term is the resource usage predicted in advance by the prediction model, which multiplied by a scaling coefficient (s) ( s > 1) in order to ensure that the system always allocate resources more than required. The second term is the total estimation of SLA violation rate within the most recent period of time with length l. If the SLA violation rate is increased ( or decreased), the amount number of resources is increased (or decreased) accordingly. By that way the number of resources is adjusted according to the SLA violation rate. After doing the above process, the number of each resource type is used in below equation to allocate the optimal number of VMs. In this study, they assume that the all VM are homogeneous and has the same capacity. For example, VM that has a capacity of 4 CPU core, 4GB memory and 1024 Mbps for the bandwidth. Therefore, these capacities are used in the below equation to compute the number of VMs needed by the application.



As for the evaluation, the proposed solution is tested on a real workload dataset generated from Google data center. They select a job that is part of a long running jobs. The selected job comprise of roughly 6000 task running within 20 days period. With respect to the performance metrics, they used CPU and memory usage to allocate resources. The proposed prediction model is evaluated against other prediction models to predict CPU and memory utilization given different window sizes (p). The prediction accuracy is measured using the Mean Absolute error (MAE) performance measurement. The results have shown that the proposed methods achieve the minimum MAE comparing to the other prediction models. On the other hand, to verify the scaling decision, they have proposed a new metric to measure the level of SLA violation Time Percentage (SLAVTP), which defined as following:



The term T under provision refers to the total time in which at least one of the allocated resources cause under provisioning (i.e. no enough resources to handle the workload), and term T execution define the total execution time of the application. The achieved results prove the effectiveness of the scaling decision in achieving the minimum SLA violation.

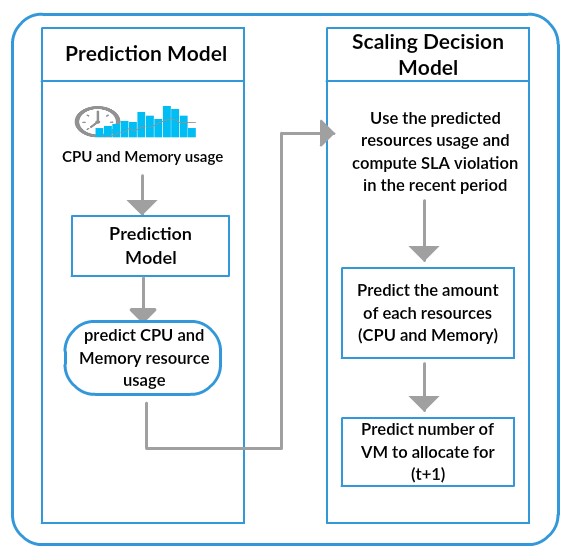


Figure 4 Proposed proactive scaling model for clouds[9]

As mentioned earlier, the accuracy of any scaling decision algorithm is affected by which time series prediction algorithm is chosen to predict the workload. Therefore, in here we discuss briefly one of the previous work [10] which focus mainly on how to improve time series workload prediction techniques given different type of workload patterns. The main objective of this work is to validate the following hypotheses: *“Prediction accuracy of predictive auto-scaling systems can be increased by choosing the appropriate time series prediction algorithm based on the incoming workload”*. [10]. To prove this hypothesis, they take the advantage of both machine learning techniques, SVM and Neural Networks (NN), to implement the workload prediction model. The novelty of this study is by building an adaptive workload prediction suite which choses automatically the most suitable prediction model, either SVM or NN, based

on the incoming workload pattern. In this study they focus on three types of workload which are: growing workload, periodic workload and unpredicted workload. The workload considered in this is study is the number of user requests per unit time. As for the evaluation, the paper use TCP-W benchmark that emulates online bookstore. Also, they used Amazon EC2 experimental infrastructure to conduct different experiments. On the client side, they developed a script that responsible to generate different type of workloads. Each workload was generated for 300 minutes and the web server layer store the total number of user requests in a log file very minute. The dataset is divided into 60% for the training phase and 40% for the testing phase. The actual data has only one feature which is the number of requests per minute. The results have shown that the SVM trained model perform better with the growing and periodic workload, while NN trained model perform has better prediction with unpredicted workload.

To summarize, time series techniques improved all reactive based methods by preparing the required resources ahead of time in order to avoid the overhead caused by VMs booting. However, none of the aforementioned studies consider the VM types. Most of the proposed work in the literature as [7] and [9], assume that all VMs are homogeneous and have the same capacity . By considering the type of VMs, it might help in building more sophisticated cost aware auto-scaler. In addition to that, only few studies in the literature address how to deal with the unpredicted load (i.e. spike workload), as discussed in [7].

# Evaluation of Techniques

In this section, we compare and contrast the different techniques discussed above. We discuss where each of the techniques could effectively be used. As for the techniques discussed above, threshold is no longer recommended or used on its own. Most of the cloud providers prefer predictive techniques. In case threshold is used it is usually combined with another technique like what the authors of [3] did to overcome some of the threshold based technique such as the fixed threshold. Furthermore, as [1] mentioned control theory is gaining more and more interest from the researchers, other than [3] who implemented the adaptive control technique, [11] also adapted a model predictive controller to propose an even triggering scaling process. Both techniques showed promising results, and shows that control theory can be used to do reactive or predictive scaling.

As mentioned earlier, due to the dynamic demands of web applications, defining fixed thresholds is a challenging task. Therefore, proactive auto-scaling techniques that don’t involve any human supervision have started gaining popularity nowadays. Reinforcement Learning (RL) and Time Series based techniques are one of the common examples. The main drawback of RL techniques is that the scaling decisions are improved on while the application runs. Thus, it takes a long time to learn and improve the results. On the other hand, the auto-scaling techniques which based on time series analysis, use historical data as prior knowledge to train the prediction model only once and then take the allocation decision accordingly. Therefore, it can be clearly seen that reinforcement learning methods can be used in the context of application which can tolerate the poor decisions made during the early stages of the learning process, otherwise Time Series Analysis based techniques can be considered.

# Conclusion

In summary, we discussed the various techniques proposed in the literature to solve the scalability problem in cloud computing. We have also pointed out the strengths and weakness of the different techniques. While evaluating and comparing one technique with another, we have shed some light on where each of the techniques would likely be better suited to be used. From our research and the papers reviewed above, we have noticed that recent works are now focusing more on the predictive techniques rather than the reactive techniques. This, we argue, is because predictive methods facilitate a more faster solution to the scalability process, by predicting the load beforehand and reacting accordingly by providing the needed resources. By doing so, these auto-scalers thus avoiding a stage where the system is either overloaded or underloaded.

# References

1. T. Lorido-Botran, J. Miguel-Alonso, and J. A. Lozano, “A Review of Auto-scaling Techniques for Elastic Applications in Cloud Environments,” *Journal of Grid Computing*, vol. 12, no. 4, pp. 559–592, 2014.
2. M. Z. Hasan, E. Magana, A. Clemm, L. Tucker, and S. L. D. Gudreddi, “Integrated and autonomic cloud resource scaling,” *Proceedings of the 2012 IEEE Network Operations and Management Symposium, NOMS 2012*, pp. 1327–1334, 2012.
3. M. Moghaddassian and A. Leon-garcia, “Adaptive Auto-scaling for Virtual Resources in Software-Defined Infrastructure,” no. 9, pp. 548–551, 2017.
4. G. Huang, S. Wang, M. Zhang, Y. Li, Z. Qian, Y. Chen, and S. Zhang, “Auto scaling virtual machines for web applications with queueing theory,” *2016 3rd International Conference on Systems and Informatics, ICSAI 2016*, no. Icsai, pp. 433–438, 2017.
5. H. Arabnejad, C. Pahl, P. Jamshidi, and G. Estrada, “A comparison of reinforcement learning techniques for fuzzy cloud auto-scaling,” *Proceedings - 2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGRID 2017*, pp. 64– 73, 2017.
6. W. Iqbal, M. N. Dailey, and D. Carrera, “Unsupervised Learning of Dynamic Resource Provisioning Policies for Cloud-Hosted Multitier Web Applications,” *IEEE SYSTEMS JOURNAL*, vol. 10, no. 4, pp. 1435–1446, 2016.
7. S. Wu, B. Li, X. Wang, and H. Jin, “Hybridscaler: Handling bursting workload for multitier web applications in cloud,” in *Parallel and Distributed Computing (ISPDC), 2016 15th International Symposium on*, pp. 141–148, IEEE, 2016.
8. G. Chen, W. He, J. Liu, S. Nath, L. Rigas, L. Xiao, and F. Zhao, “Energy-aware server provisioning and load dispatching for connection-intensive internet services.,” in *NSDI*, vol. 8, pp. 337–350, 2008.
9. D. Tran, N. Tran, G. Nguyen, and B. M. Nguyen, “A proactive cloud scaling model based on fuzzy time series and sla awareness,” *Procedia Computer Science*, vol. 108, pp. 365–374, 2017.
10. A. Y. Nikravesh, S. A. Ajila, and C.-H. Lung, “Towards an autonomic auto-scaling prediction system for cloud resource provisioning,” in *Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pp. 35–45, IEEE Press, 2015.
11. S. Cerf, M. Berekmeri, B. Robu, N. Marchand, and S. Bouchenak, “Cost function based event triggered Model Predictive Controllers application to Big Data Cloud services,” *2016 IEEE 55th Conference on Decision and Control (CDC)*, no. Cdc, pp. 1657–1662, 2016.