A Review of Auto-Scaling Techniques for Web Applications

Sara Al-Rasbi 200909464
Nada Aboueata 200900288
Rahma Ali 201001782

Abstract

The rise in the popularity of cloud computing has resulted in the need to advance techniques that enhance their performances. Cloud service providers have been competing with one another in order to deliver to the users the best solutions to overcome performance issues by using different auto-scaling techniques. Many techniques have been proposed in the literature, starting from basic methods that are based on threshold concepts to more advanced techniques that employ real-time data and machine learning techniques. These techniques aim to auto-scale the resources without any intervention from the Cloud service providers. In this paper we review papers that have implemented different auto scaling techniques.

1 Introduction

With the increase use of web service based applications, techniques to make cloud computing services more effective and efficient while maintaining minimum costs to the service providers have started to garner significant interest. One of the ways this is 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 namely infrastructure-as-a-Services (Iaas), Platform-as-a-Service (PaaS) and Software-as-a-Service (SaaS) [1]. 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 example of IaaS 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 is necessary to develop a sophisticated techniques 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. Reactive based 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. On the other hand, proactive based techniques attempt to predict the future workload and prepare the required number of resources, known as resource scaling, ahead of time. Resource scaling can either be done horizontally or vertically. Horizontal scaling means adding or removing server replicas that are running on VMs. While, 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 required functioning of the application, by providing an acceptable level of Quality of Service (QoS) measure. These could be features 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. A monitoring system is 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. After which 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 criticizes different auto-scaling techniques that are proposed by researchers in this field. In our review, we use the classification done by [1].The authors suggest to classify the papers of scalability problems on the approaches used in implementation, rather than using the well known broad categories - reactive and proactive. The rest of the paper is organized as follows: We start with the main terms that are frequently associated with scalability in web applications. Then we move on to discuss the papers starting with threshold-based techniques in section 2, queuing Theory in section 3. Then, we discuss papers that implement reinforcement Learning techniques in section 4, and time series analysis in section 5. Finally, we compare between the auto-scaling categories and discuss their strengths and weaknesses in section 6. Finally, we conclude this review in section 7.

2 Threshold-based Scalability Technique

Threshold-based rules are techniques that were very popular with cloud providers because of their simplicity that revolves around defining policies and rules which trigger the scaling process. However, this approach is very sensitive to the defined thresholds, as it might lead to the wrong scaling decision if they were not set accurately. Furthermore, the crafting of rules require a good understanding of the application workload and its trends. As in any rule, a rule consist of a condition and a consequence/action. In scalability problems, the condition usually include the performance metrics. Typically each metric could have an upper threshold and a lower one. If any of the thresholds is met, it means that a scaling action might be needed [1].

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

For instance, Hasan et.al [2], added extra thresholds and a duration period to monitor the performance metrics behavior. If we look at the scaling up method, they add an extra threshold thats a little lower than the upper threshold. If a performance metric passes this threshold, the observation period starts. The scaling only happens if the value of the metric remains between the two thresholds. In another effort to enhance threshold, The authors of [3] make

use of threshold-based techniques performance in accuracy and precision without having to set fixed thresholds. Thus, proposing an adaptive method to change the threshold based on real-time data from the system current state. They proposed two methods, one for scaling up and the other for scaling down. 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. The authors do not propose an allocation algorithm as it is out of the scope of their goal.

Data is regularly collected from sources like CPU or memory. Then they are fed into a data analysis component that decides whether to scale or not. In the algorithm, they have two parameters α and β. The parameter α is used to increase or decrease the initial threshold T, if it is scaling up or down. Whereas, β determines the monitoring period when the threshold is reached.

The Figure 1 shows the steps of the scaling up algorithm, after analyzing the collected data, if the threshold T is met the analyzer will monitor the system state for β seconds. If the value of the VM collected data - CPU or memory utilization - is still above the threshold, the value of the threshold is increased by α percent. Otherwise, if 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 $(100-\alpha)\%$, once the threshold T reaches a point where $T >= (100 - \alpha)\%$, the scaling up is triggered. In this algorithm the selection of the initial threshold and α is critical, they both need to be selected carefully so that the system does not reach the full utilization threshold in the first run of the algorithm.

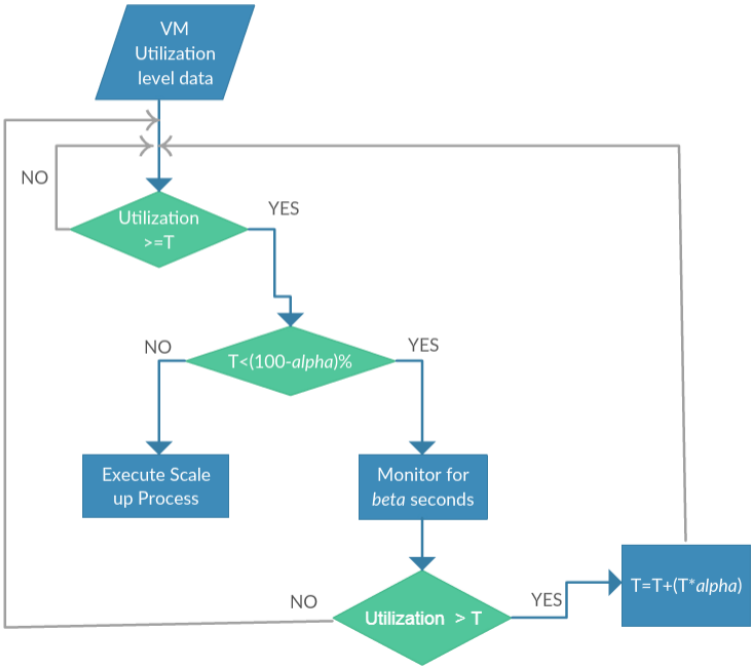

Figure 1: The Scale up Detection.

In case of scaling down, the authors define a parameter called $Tmin$ that indicates the minimum tolerance level that the threshold T can decrease to. Similar to the method of scaling up, once the utilization of the VM resources decreases to the point of 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. After that, the method continues the observation for another β second. If by the end of this period if: $Tmin < VMutlization < T$, then the changes are acknowledged, 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 scaling is triggered. They use a “Ripeness Function" (RF), for situations where in the second iteration VM utilization might exceed the threshold suddenly. 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, by using the RF they force an extra observation for another β seconds. In this period, the value of the threshold will be updated and acknowledged; if and only if, the captured value of the resources utilization level at the end of the β seconds interval is the same value at the start of the interval.

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 with varying number of users connecting to the VM. They evaluated their algorithm against a simple threshold-based algorithm. The results of the evaluation showed that their proposed algorithm outperforms the basic threshold algorithm. the threshold based algorithm was deciding scaling actions as soon as the thresholds were met. in most of the cases the decisions were haste and unneeded, meanwhile, the adaptive threshold approach monitored the system state for awhile and only signaled for scaling after confirming the system requires scaling.

As mentioned earlier, 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 performance of this method relies on the initial threshold, and α .So,if they were poorly selected this algorithm will fail to scale correctly.

3 Queuing Theory based Scalability Technique

The arrival of request to web applications, and its associated information like waiting time, queue length, distribution of the request arrivals can be useful to use in scaling web applications. Queuing theory is a study of queues in mathematics. In a typical queue, request come to the system at an average arrival timeλ, then each request is enqueued until it is serviced. A single queue can be serviced by a server or different servers[1]. Huang et. al. [4] employed queuing theory to achieve auto-scaling in web applications. The motivation behind their 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 it oscillate over time; which could lead to a phenomenon referred to as peak-valley 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."*. In queuing theory, the notation used to describe the system is written as follows A/B/C/X/Y/Z, where:

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 queuing model the authors adapt is shown in Figure 2, and 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 (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 using the performance metrics of the system in a time t; such as idle probability of the system, average

number of requests in a queue, average number of requests 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 of resources, and the waiting time in the queue. In this paper the authors refer to resizing a VM- size of CPU or memory- or adding VMs as resources and to facilitate the calculations they consider resources as an integer. The proposed resource allocation algorithm computes the optimal number of resources (V_{t+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 in the system(Ws), thus the objective function will be:

$$minf(Vt+1) = V_t + 1 + \alpha.Ws(V_t + 1)$$

As for the auto-scaling of VMs, an algorithm determines if the system should be scaled up or down, and how much to scale. The first step is to find the optimal number of resources using the resource allocation algorithm. Then if optimal resources V_{t+1} is greater than the current resources V, the system needs to be scaled up; otherwise, if $V_{t+1} < Vt$ the system needs to scale down.

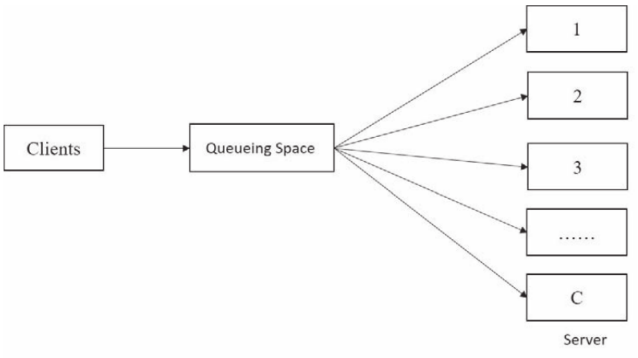

Figure 2: The Queuing Model M/M/C.

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, A is the total of resources remaining for each VM and B is the maximum number of resources the VM supports,if A<B then the VM can be scaled up. To scale, the average of A is added to the VM and A is updated with the remaining resources of the VM.

As for scaling down, they only consider horizontal scaling because 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 is to select which VMs to close while keeping the resources greater than $V - V_{t+1}$.

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 whose resources are 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. The proposed system terminates 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 servers is viewed as a queue and made use of its simplicity to craft solutions for the auto-scalability problem. However, a queuing theory solution cant be applicable in all situation because queue that based on exponential distribution must have a variant coefficient of one, otherwise, the queuing theory wont be applicable to use.

4 Reinforcement Learning based Scalability Technique

In this section and next one, we will focus on proactive based auto-scaling techniques. Reinforcement learning (RL) technique is one such method that is used in proactive based auto scaling techniques. In Reinforcement learning technique, the automation of the scaling task is not dependent on any pre-existing knowledge or any performance model of the application [1]. In Reinforcement learning technique, scaling decisions are made and improved on while the application is running. The auto scaler component in Reinforcement learning technique 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, scale vertically, or perform no action at all. These actions are selected based on the current state of the application, its workload and the response rate. In Reinforcement learning technique, 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 a 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 the given application states. Intuitively, the agent learns and assigns actions to different states in such a way so as to maximize 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. In order to understand how Reinforcement learning techniques are applied in the literature, for this review, we reviewed the implementation of Reinforcement learning techniques in both a single layered application and in a multi-layered web applications.

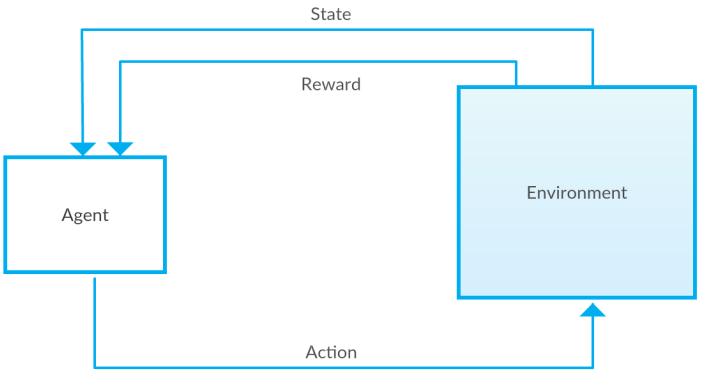

Figure 3: Reinforcement Learning Architecture.

Arabnejad et al in their paper [5] address utilization of Reinforcement learning technique based auto scaler in a single tier application. They extended a previous work that uses fuzzy logic to auto scale and combined it with two Reinforcement learning techniques - Q-learning and SARSA - to present two different auto scaling models. In their work, they made the process of allocating resources dynamic by applying two Reinforcement learning techniques to the output of a fuzzy logic auto-scaling controller. The output of the controller is the predicted number of required resources. They then compared these two proposed models in terms of meeting QoS requirements in a cost-effective manner. Q-learning and SARSA, the techniques used, 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 set of best possible next states making it an off-policy technique, SARSA on the other hand compares the current one with the actual next state obtained from pre-defined policies, making it 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. Which in theory would mean that the learning process would be faster.

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 the number of virtual machines utilized. 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 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 here 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 the literature, the learning process is implemented in different ways. for instance, one way it is done is by first mapping all system states to all possible actions. The aim of the agent, in this scenario is to then find a policy that maps every state in the application to its respective best action [1]. They used this approach in their proposed solution. 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, thus the higher the q-value assigned to the state-action pair. Here, the q-values are initially all set to 0. As learning progresses, these values are updated. In the case of Fuzzy SARSA Learning (FSL), the new state is selected by considering the actions available to be chosen from and selecting the action with the maximum q-value obtained from following the policy. 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 without following the policy. Once a new state is reached, the current count of vms and response time of the state are checked. Using these two metrics, the reward is calculated. Two other measures - the number of resources obtained - which regulates the cost - and the SLO violations committed are considered in this calculation.

In order to evaluate their work, they implemented their proposed models on OpenStack and simulated different types of workload patterns.The three types of representative workload patterns they evaluated their work on are the predictable bursting pattern, the Variations pattern and the on-and-off pattern.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 in 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 on-policy model and thus making the learning and scaling process faster that makes it perform poorly 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 terms of evaluating their proposed solutions 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.

Iqbal et. Al in their paper [6] attempted to address auto scaling using Reinforcement learning technique for multi-layer web application. Their proposed solution is a complete auto scaler that has both workload pattern prediction and resource provisioning policy learning. In the policy learning part of their proposed solution, Iqbal et. al. proposed an implementation that performs

adaptive resource allocation of multi-tier Web applications using an online unsupervised method for policy learning. They set their learning agent to use a simplistic method to find the policies that maximize an objective function which rewards satisfying a response time adherence, SLA contracts with minimal resources utilization as possible. They defined the system state at time t as

$$s_t = (U, P(C), \lambda, p)$$

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, . . . , u_n), where u_i indicates the number of machines allocated to tier i in the application. P(C) is the current workload pattern of the application, λ is the arrival rate of the requests and p is the 95th percentile of the service time for the application. These measurements are taken for a known time period. Their auto scaler was designed to only do vertical scaling but on different tiers. So according to need, the agent could scale up the Web tier (a_w), scale up the database tier (a_d), scale up both tiers (a_b), or do no scaling (a_ϕ). Thus, the set of possible actions that the agent could perform can be shown as

$$A = a_w, a_d, a_b, a_\phi$$

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, they implemented their approach in 2 algorithms, one for the exploration phase, which is the online learning phase and other for the exploitation phase which is the decision-making phase. The resources scaling process 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. The process of 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 arises.

In order to evaluate their work, 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 modelled each user session to have 32 user requests and assigning lesser resources than required to complete their tasks. In the case of the heavy workload experiment, they modelled each user session to have 10 user requests that require intensive resources which 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 was that, since the workload is always increasing in both experiments, the other methods fall into the mistake of overprovisioning of resources so as to avoid increasing the number of requests that violate the SLA. However, this, they argue 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 at provisioning resources. The basic guideline of the reward function is to encourage the learning agent when it respects the SLO while utilizing the 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. They argue that scaling the database tier introduces overhead of load balancing at the web tier and data synchronization in the database tier which can be avoided by scaling at the web tier.

While this paper offers compelling arguments, it also has weaknesses. One clear weaknesses 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.

5 Time Series Analysis based Scalability Technique

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, a significant portion of research in the literature have addressed this issue using Time series analysis techniques. An auto-scaling problem can be divided into two main steps: a prediction step that predicts the future values of performance metrics and decision-making step for resources allocation.

Time series analysis can only be applied to the first step, which is the prediction phase. Based on this prediction, the second step allocates resources accordingly. In this section, we discuss studies that propose resource allocation techniques that rely on time series prediction methods in order to predict the future values of performance metrics.

Generally, time series techniques predict future values by identifying repeating patterns or by analyzing the historical data. In the context of auto-scaling, data are performance metrics (such as CPU utilization, memory usage, requests arrival rate, and others) which 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 has been applied mainly to predict the workload or resources utilization. It can be classified into two main groups: techniques that focus on direct prediction of future values, or approaches that extract pattern followed by the time series (if present) and then extrapolate it to predict future values. Averaging methods, Auto-regression, Auto-regression Moving Average, and different machine learning approaches are some of the methods the belong to the first group [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 earlier observation (i.e. Weighted moving Average) [1].

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 both, horizontal and vertical scaling since each method has its strengths and weaknesses. 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 because they predict the workload ahead of time and decide on the needed resources before violating any SLA rules or standards. Despite its success, proactive horizontal scaling still causes extra cost and significant overhead specifically with the short-term bursting workload. In reality, a short-term bursting workload is hard to predict, and it requires adding or removing VMs very frequently which leads to significant overhead and extra cost. Therefore, proactive horizontal scaling is not suitable for frequent short-term allocation. The authors addressed this problem by using a lightweight reactive vertical scaling approach with the short-term workload, in which it adds/removes VCPUs and RAMs instantly.

As shown in Figure 4, 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 (W_i) and VM in that tier (U_{vm}). They used the Sparse Periodic Auto-Regression [8] which is a simple time series analysis method, to predict the workload. After that, they use below equations to calculate the required number of VCPU (I_{vcpu}) and VMs (I_{num}) either to scale horizontally or vertically.

$$I_{num} = \left\lceil \frac{W_i}{U_{vm}.\alpha_{num}} \right\rceil$$

$$I_{vcpu} = \left\lceil \frac{U_{vm}}{U_{vcpu}.\beta_{num}} \right\rceil$$

Where U_{vm} , $Uvcpu$ shows the predicted workload pressure per 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 a SLA violation and the system reacts by taking a vertical scaling action.

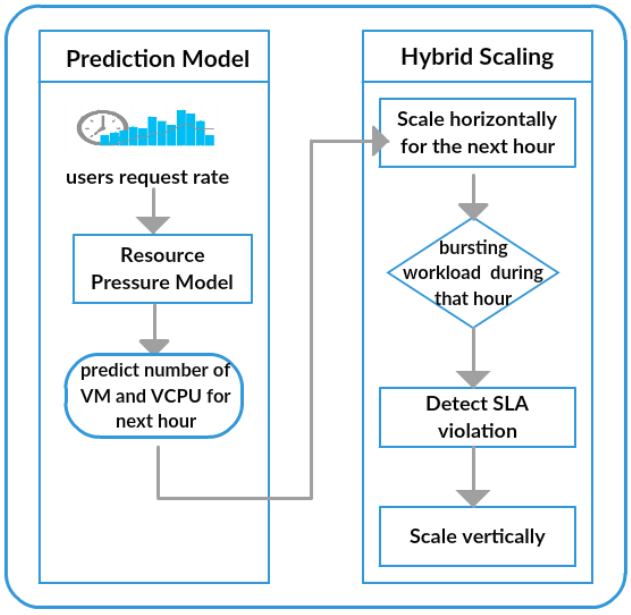

Figure 4: The HybridScaler architecture

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 effectiveness and efficiency of the proposed algorithm and the baseline algorithms. 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 showed that the proposed algorithm outperforms the baseline by reducing the average response time by 16-39% and SLA violations by 24-50%. 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. So, in 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 uses fuzzy approach to transform the monitored performance metrics into a fuzzy time series, which later used as an 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 be defuzzified into real values, which are the final output of the prediction model. As shown in Figure 5, 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 number of each resource type i ($r_i{}^{\text{decis}}(t+1)$), such as number of CPU and Memory resources, is calculated using the equation below:

$$r_i{}^{\text{decis}}(t+1) = s * r_i{}^{\text{pred}}(t+1) + \frac{1}{l} * \sum_{z=t-l+1}^{t} r_{violation}(z)$$

Calculating the 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 is multiplied by a scaling coefficient (s) (s $>$ 1) to ensure that the system always allocates more resources 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 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 the below equation to allocate the optimal number of VMs. In this study, they assume that all VMs are homogeneous and have the same capacity. For example, VM that has a capacity of 4 CPU core, 4GB memory and 1024 Mbps for the bandwidth. These capacities are used in the below equation to compute the number of VMs needed by the application.

$$n_{VM}{}^{\text{alloc}}(t+1) = max\{\ \frac{r_1{}^{\text{decis}}(t+1)}{C_1}, \frac{r_2{}^{\text{decis}}(t+1)}{C_2}, ..., \frac{r_M{}^{\text{decis}}(t+1)}{C_M}\}$$

To evaluate the model, the proposed solution was tested on a real workload dataset generated from Google data center. They select a long running job. The selected job comprises of roughly 6000 task running within a period of 20 days. With respect to the performance metrics, they used CPU and memory usage to allocate resources. The proposed prediction model was 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 compared to the other prediction models. To verify the scaling decision,

they proposed a new metric to measure the level of SLA violation Time Percentage (SLAVTP), which is defined as following:

$$SLAVTP = \frac{T_{under-provisioning}}{T_{execution}}$$

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 defines the total execution time of the application. The achieved results prove the effectiveness of the proposed method by achieving the minimum SLA violation.

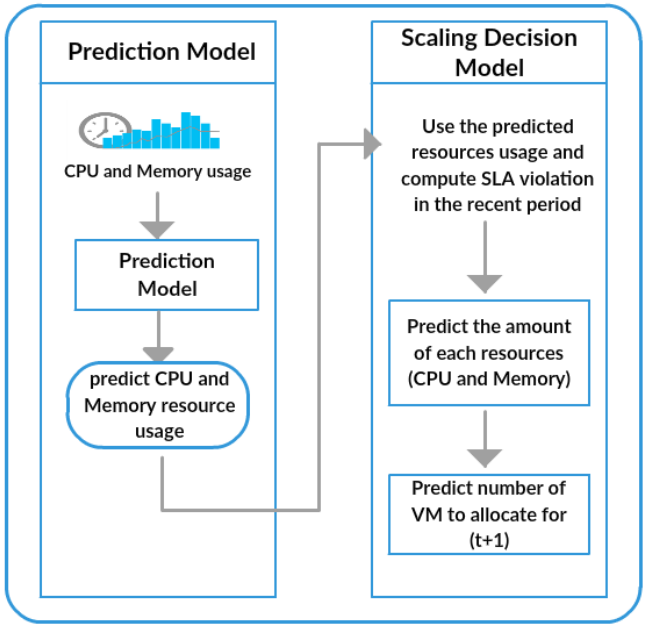

Figure 5: Proposed proactive scaling model for clouds[9]

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 focuses mainly on how to improve time series workload prediction techniques given different types 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 chooses 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 unpredictable workload. The workload considered in this study is the number of user requests per unit time. In the evaluation, the authors 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 thats responsible for generating different types of workloads. Each workload was generated for 300 minutes and the web server layer stored the total number of user requests in a log file every minute. The dataset was 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 performs better with the growing and periodic workload, while NN trained model has better prediction with unpredictable 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 time. 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].

6 Evaluation of Techniques

In this section, we discuss the strengths and weaknesses of the different techniques discussed above. We also discuss where each of these techniques could effectively be used.

The techniques discussed in sections 2 and 3 which are threshold-based and queuing theory techniques are reactive based techniques. In general, such techniques have a problem of having overhead delays in the auto scaling process. This is caused due to the fact that allocated resources need to be setup at the time of need. For example, when more virtual machines are required, booting them usually takes anywhere from 5 to 10 minutes thus resulting in a noticeably slow reaction to workload changes [1]. Moreover, auto-scalers based on only threshold rules are no longer the trend, even when they are used, they are done in collaboration with other techniques to overcome their weaknesses as seen in the work of Moghaddassian et.al [3]. Queuing theory, although comparatively more favourable due of its simplicity, it is still not applicable for all systems. This is because its use is bound by the nature of the requests arrival rate in the system.

Proactive based techniques such as reinforcement learning and time series analysis techniques as decribed in sections 4 and 5 came to solve the problem of overhead delay caused due to resources setup time. They do this by predicting the need for the resources before hand and thus resulting in making them quickly available when needed. These are especially useful for the dynamic demands of web applications, since these techniques are more dynamic in nature than the former. The technique of reinforcement learning technique is especially appealing due to its ability to easily be plugged and used in any application on the go. As explained earlier, it adopts a learning process and auto scales accordingly. This is useful for applications that have unpredictable workload patterns. On the other hand, for applications that have predictable workload patterns and stored application logs and usage details, time series analysis technique seems like an appropriate choice of auto scaler technique. This is because time series analysis based techniques use historical data as prior knowledge to train the prediction model only once and then take the allocation decision accordingly. However, despite the success of proactive based methods, they nevertheless have their own short-comings. In the case of Reinforcement learning technique, one of its major drawbacks is that its learning process is in most cases a time-consuming process. This is caused due to the time it takes for the learning process to learn the most appropriate action for each state. Due to the fact that this auto-scaling technique is done on the fly using a trial and error method as mentioned earlier, the learning process may become impractically long in some cases. In general, these techniques lack at performing seamlessly for very unpredictable types of workload such as those brought by short-term burst workload. Consequently, there is a lot of research currently being done in the literature to try to improve proactive approaches so as to handle this type of workloads.

7 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 weaknesses 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, we have noticed that recent works are now focusing more on the predictive techniques rather than the reactive techniques. This, we argue, is because proactive 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, the proactive based auto-scalers are better at avoiding overprovisioning or underprovisioning of resources.

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.