Auto-Scalability of Web Applications

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

November 9, 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.

include the abstract

1 Introduction

Nowadays, due to the dynamic demands of web applications, cloud computing technology became an emerging need to meet those demands. Due to its elasticity nature, users can acquire and release resources as needed, which makes the cloud computing technology more and more popular nowadays. The industry environment uses cloud computing for different purposes, for instance, hosting web applications, storage and backup or for running batch jobs. The three main markets associated to cloud computing are [1] :

1. Infrastructure-as-a-Services (IaaS) : The cloud provider provision all infrastructure components needed for server deployments. These can include the networks resources such as the processing, storage, bandwidth requirements and others. One of the most common examples is Amazon EC2.

2. Platform-as-a-Service (PaaS): This one is at higher level than the IaaS, typically it includes all the programming environments (i.e. platforms and tools), on which the cloud users can develop and deploy their application. Examples of PaaS such as Google App Engine and Microsoft Windows Azure

3. Software-as-a-Service (PaaS): It mainly designates the host vendor applications such as Google Apps and Microsoft office 365.

As mentioned earlier, elasticity is one of the main key characteristics of cloud computing. Even though users can allocate (add/remove) resources dynamically according to the current demands, however, it's still challenging to decide the right number of resources. Therefore, it's necessary to develop a sophisticated technique for resource allocation according to the application demand, which known as auto-scaling techniques. Auto-scaling techniques are divided into two group: reactive and proactive based. The reactive based techniques are based on a set of predefined rules, in which the system reacts to workload changes instantly only when those changes have been detected. The main issue with reactive approach, is that allocating VM resources instantly might cause overhead of VM booting (Usually VM booting time range from 10 to 15 minutes). On the other hand, the proactive techniques try to predict the future workload to prepare the required number of resources to handle the workload. By this way the VM booting overhead drawback can be eliminated by the proactive approaches. Despite its success, it still lacks at predicting the unpredicted workload such as the short-term spike workload. Therefore, many studies in the literature try to improve proactive approaches to handle this type of short-term spike workloads.

Resource scaling can be either horizontal or vertical scaling. Horizontal scaling (scale out/in) refers to add/remove server replica that's running on vertical machine (VM). On the other hand, vertical scaling refers to changing number of resources (increase/decrease) allocated to an already running virtual machine which includes CPU, Memory and others.

Any auto-scaler must guarantee the correct functioning of the application, by providing an acceptable Quality of Service (QoS) metrics such as response time, high throughput, and high availability to the end users. Without such assurance, the Service level agreement (SLA) between the customer and the service providers will be lost. Typically maintaing the SLA, depends on the degree of satisfying those QoS properties. In any case, the auto-scaler should be able to address the trade-off between minimizing the cost of allocating resources and maininting the SLA by assuring the highest QoS propoerties.

Generally, any auto scaler system match the MAPE loop autonomous, which consists of four main phases: Monitoring, Analysis, Planning, and Execution. In the context of auto-scaling, first a monitoring system will be responsible to gather historical data about the system this includes performance metrics such as hardware counters and application log information. Thereafter, the Analyzer model analysis the retrieved data to predict the future values, and then the planning model will plan for the suitable resources allocation decision. Finally, the provider execute the plan of auto-scaler accordingly [1].

In this paper we discuss 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 based on the nature of the techniques to solve the scalability problem, rather than use the overused categories reactive and predictive. Therefore, the rest of this paper is organized as follows, section 2 includes the classification techniques suggest by [1]. we start with Threshold-based rules in section2.1, Control Theory in section 2.2, Queuing Theory in section 2.3. section 2.4 discuss Reinforcement Learning techniques, lastly Time Series Analysis in section 2.5. Later on, we compare the different techniques and discuss their strength and weakness in section 3. Then we conclude in section 4

2 Classification of Scalability Techniques

2.1 Threshold-based Rules

Threshold-based rules are techniques that are very popular with cloud providers. It is a very simple method that defines policies and rules that trigger the scaling process. However, this approach depends on the thresholds define 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. In the MAPE loop, threshold-based approaches are considered a method in the planning phase. As in any rule, rules have two parts: a condition and a consequence or an action. The 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. Some condition also includes a duration where an observation is done to monitor for how long a condition is met, once the duration is over the action is executed. Furthermore, in horizontal scaling, the number of resources to be added or removed should be defined and fixed beforehand [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 are isolated during computations. Furthermore, the authors discuss the need to also scale network resources in relation to the other scaled resources, like the need to scale the firewalls or the load balancer when virtual compute resources are scaled. As an input the system is provided, 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. Like grouping by resource type or grouping resources according to whom control they are under;such as, a firewall, a load balancer, etc. the last inputs the system needs are the threshold policy and the auto-scaling policies, Both acquisition and release of resources. 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.

2.2 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 resources are added or removed. 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 -in terms of 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 until a desirable system output is achieved. 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.

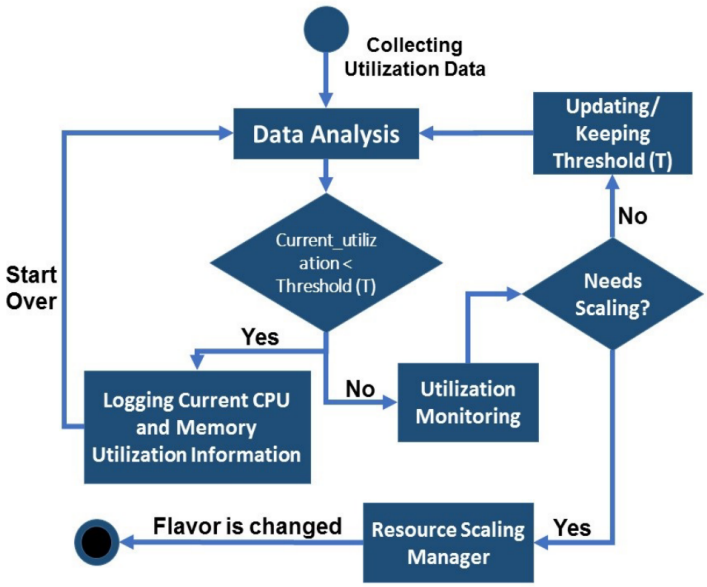

Figure 1: The Decision making in the proposed method.

In the work of [3]. They 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 immediately after a threshold is met, but they 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. Both parameters could be either fixed or dynamically changed, and this choice is left to the user implementation of the method. The algorithm works under the assumption that the current state of the system is getting worse. Thus it waits for β seconds to observe until the assumption is proven correct.

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 be decreased. 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 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.

2.3 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 timeλ, 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 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."* 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)

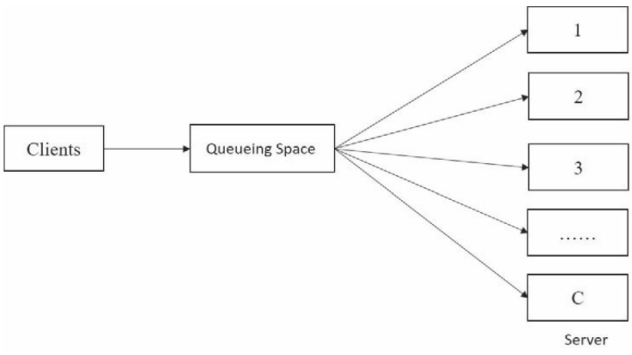

Figure 2: The Queuing Model M/M/C.

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 auto-scaling 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 (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 the system(Ws) thus the objective function will be:

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

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 V_{t+1} if its greater than the current resources V then the system needs to be scaled up; otherwise, if $V_{t+1} < Vt$ 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 $V - V_{t+1}$.

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 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.

2.4 Reinforcement Learning

Reinforcement learning (RL) is another technique used in auto scaling. Reinforcement learning 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 application 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 or performance. In RL, the action chosen for a given state is 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 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 can be implemented is by first mapping all system states to all possible actions. The aim of the agent 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 higher the reward obtained for an action performed on a state, the higher the q-value assigned to that pair. Initially the q-values are all set to the same value. As learning progresses, these values are then changed. Because a greedy approach is followed in selecting the appropriate actions, actions belonging to pairs with higher q-values are selected the next time a particular state is reached. While this sounds like an advantage in that it can easily be plugged and used in any application on the go, it had its disadvantages in 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. This reaction is usually done on the fly using a trial and error method making the learning process unpractically long in some cases. As explained earlier,

there are three types of workload patterns namely the Predictable Bursting pattern, the Variations pattern and the ON/Off pattern. The authors tested bth 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. As a result, SARSA has a better tendency to get a more accurate number of VMs launched in such scenarios. However, the opposite is the case for the Variations workload pattern. FQL is performs significantly better than the FSL approach. Interestingly it is due to the same reason of FSL being an on policy model and thus learning and scaling faster that makes it perform badly in scenarios where there is variations in workload resulting in random fluctuations and non-periodic behaviour. In the case of the ON and OFF workload patterns, both of the models perform similarly.

2.5 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 doesn't react fast enough to allocate the right number of resource at the right time. Therefore, many of the studies 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 phase that predict future values of performance metrics and then the decision-making phase for allocating resources. Time series analysis can be only be applied to the first part, 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 resource allocation. In this review, we discuss both type 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 present) 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 resource usage. 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. The averaging methods calculates the future values in different ways, either by treating all the 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 [5], a hybridScaler algorithm is proposed 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 VM machines. 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 resource 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 reallocation. The paper addressed this problem by using a lightweight reactive vertical scaling approach with short-term workload, which hots add/removes VCPUs and RAM timely.

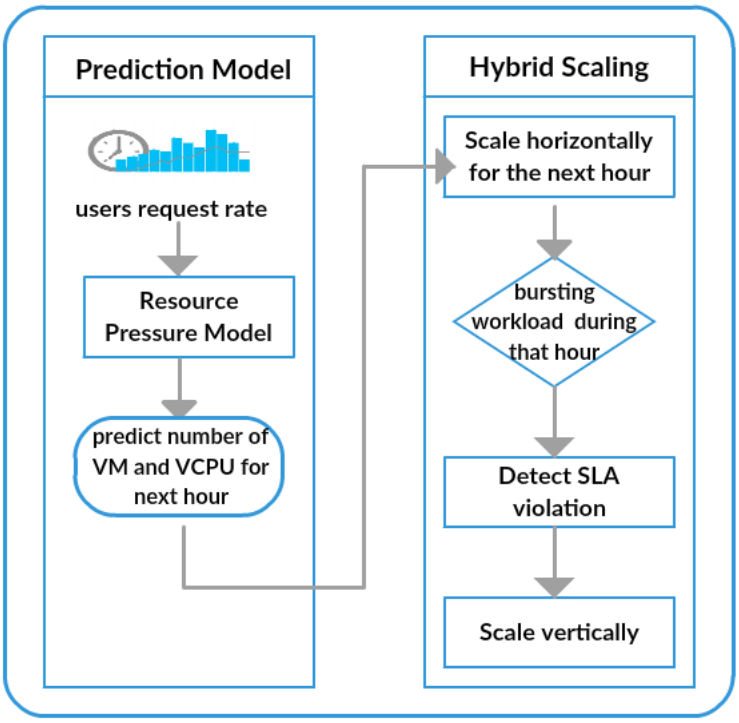

Figure 3: The HybridScaler architecture

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 for the next hour (VM or unit resources as VCPU) to allocate for multi-tier application. This is done by predicting the workload pressure per each tier and per each resource in that tier when number of resources is Ivm or Ivcpu (VM and VCPU). They used the Sparse Periodic Auto-Regression [6] which is a simple time series analysis method, to predict the workload. Thereafter, they use below equations to calculate the required number of VCPU and VM resources either to scale horizontally or vertically.

Equations

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, RUBiS. The workload was generated by using the ClarkNet trace which has two weeks access logs. They used only the request rates per minute for the first week and then average all rates per hour to train the proposed prediction model (i.e. resource-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 SLO 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 SLO violations rate. In addition, it keeps a table CPU utilization between 60% to 70%, to avoid any resource wastage.

Similarly, to the previous work [5], in [7] they 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

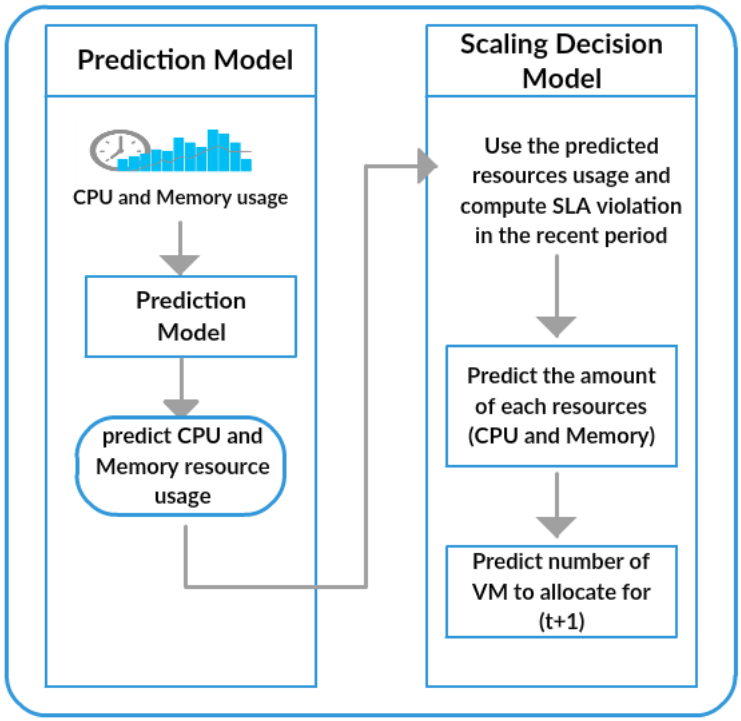

Figure 4: Proposed proactive scaling model for clouds [7]

that uses the predicted values and 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 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 machine learning algorithm. 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 defuzzified into real value, which are the final step 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 resources allocation. To perform the scaling action, the amount number of each resource type (i) (such as CPU, RAM, and bandwidth) is calculated using equation (?), which in turn will be used to allocate the proper number of VM using equation (?).

As shown in equation (?), calculating the amount number of each resource type is based on two terms. The first term is the resource usages 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 the demand. 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 output of each resource type is used in equation (?) to allocate the optimal number of VM. 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 equation (?) to compute the number of VM 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 in equation (?). 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.

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 [8] 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" [8]. 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 benchmarks 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.

As mentioned earlier, time series techniques improved all reactive based techniques by preparing the required resources ahead of time to avoid the overhead of VM booting .However, none of the aforementioned techniques consider the VM types. Most of the prosed work in the literature assume that all VMs have the same capacity and homogeneous. By considering this factor, 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).

3 Evaluation of Techniques

In this section we explain the evaluation techniques used by the papers in order to evaluate their proposed solutions. One way to evaluate the performance of an auto scaler, workload patterns need to be manipulated. Application workload patterns can be categorized in three representative patterns [19]: (a) the Predictable Bursting pattern indicates the type of workload that is subject to periodic peaks and valleys typical for services with seasonality trends or high performance computing, (b) the Variations pattern reflects applications such as News and Media, event registration or rapid fire sales, and (c) the ON and OFF pattern reflects applications such as analytics, bank/tax agencies and test environments

write evaluation

1. Predictable Bursting 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.

2. Variations pattern reflects - where the spike of the user requests is not predictable and happens in a random fashion. Usually seen in news applications and event registration applications.

3. ON and OFF pattern - This pattern is the most stable of the workload patterns and reflects applications that have a well known and fixed usage period, such as test environments.

4 Conclusion

In Summary, we discussed the various techniques proposed in the literature to solve the scalability problem in cloud computing. Different approaches provide different advantages and disadvantages, and as noticed recent works are focusing more on the predictive techniques rather than the reactive techniques. The predictive methods facilitates the scalability process, as they attempt to predict the load and react by providing the needed resources thus avoiding a stage where the system is overloaded.

write conclusion

Todo list

4.1 How to add Tables

Use the table and tabular commands for basic tables — see Table **??**, for example.

4.2 How to write Mathematics

LaTeX is great at typesetting mathematics. Let $X_1, X_2, \ldots, X_n$ be a sequence of independent and identically distributed random variables with $\mathrm{E}[X_i] = \mu$ and $\mathrm{Var}[X_i] = \sigma^2 < \infty$, and let

$$S_n = \frac{X_1 + X_2 + \cdots + X_n}{n} = \frac{1}{n}\sum_{i}^{n} X_i$$

denote their mean. Then as n approaches infinity, the random variables $\sqrt{n}(S_n - \mu)$ converge in distribution to a normal $\mathcal{N}(0, \sigma^2)$.

4.3 How to create Sections and Subsections

Use section and subsections to organize your document. Simply use the section and subsection buttons in the toolbar to create them, and we'll handle all the formatting and numbering automatically.

4.4 How to add Lists

You can make lists with automatic numbering . . .

1. Like this,
2. and like this.

. . . or bullet points . . .

- Like this,
- and like this.

4.5 How to add Citations and a References List

You can upload a `.bib` file containing your BibTeX entries, created with JabRef; or import your Mendeley, CiteULike or Zotero library as a `.bib` file. You can then cite entries from it, like this: [**?**]. Just remember to specify a bibliography style, as well as the filename of the `.bib`.

You can find a video tutorial here to learn more about BibTeX.

We hope you find Overleaf useful, and please let us know if you have any feedback using the help menu above — or use the contact form at `https://www.overleaf.com/contact`!

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] S. Wu, B. Li, X. Wang, and H. Jin, "Hybridscaler: Handling bursting workload for multi-tier web applications in cloud," in *Parallel and Distributed Computing (ISPDC), 2016 15th International Symposium on*, pp. 141–148, IEEE, 2016.

[6] 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.

[7] 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.

[8] 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.