Adaptive Auto-scaling for Virtual Resources in Software-Defined Infrastructure

Morteza Moghaddassian
Department of Electrical and Computer Engineering, University of Toronto
ON,Canada
m.moghaddassian@utoronto.ca

Hadi Bannazadeh
Department of Electrical and Computer Engineering, University of Toronto
ON,Canada
hadi.bannazadeh@utoronto.ca

Alberto Leon-Garcia
Department of Electrical and Computer Engineering, University of Toronto
ON,Canada
alberto.leongarcia@utoronto.ca

***Abstract*—Auto-scaling is a key challenge and benefit in cloud computing infrastructures where applications are deployed on one or more virtual machines (VMs) to balance efficiency in use against delivered performance. In different scenarios, there may be a need for either horizontal or vertical scaling. Therefore, scaling is an important operation of cloud management systems. One way to enable scaling as an automated service is to use a model to predict the VM's future state as a function of time. However, this method is not completely feasible, because the performance of a VM is so dynamic and depends on many parameters. A simpler approach to enable auto-scaling is to use real-time utilization data of VM's and a set of fixed thresholds to execute scaling when thresholds are crossed. However, this method is prone to false positive decisions. Uses this paper, we propose an adaptive method that uses threshold-based mechanisms to control the auto-scaling process and leverages the accuracy and precision given by threshold-based methods to reduce the number of false positives. We present performance results, comparing the fixed threshold methods with our proposed method. We show that our method frequently correctly triggers scaling process in situations where fixed threshold based measurement methods fail.**

I. Introduction and Related Work

Cloud computing is a state-of-the-art technology to provide shared, reconfigurable, affordable and scalable storage and computing resources. Typically, users are enabled to have a full control and detailed visibility of their resources by using the advanced technologies such as virtualization. Resources are also accountable such that users are billed based on their usage or on a pay-per-use model (1). For users, cloud computing significantly reduces operational costs and enables users to focus and invest on the quality of their services. The cloud operator needs to use efficient resource management methods to support the scalability and reliability of such large infrastructures. These resource management systems require a complete visibility of the infrastructure resources, which can be obtained using network and resource monitoring tools.

Traditionally, resource monitoring tools such as Ganglia (2), OpenNMS, Nagios, ZABBIX (3), FlowSense (4) and PayLee (5) were used to collect and visualize data from network and computing resources. However, the large networks in clouds are much larger and more complicated than previous networks. Therefore, as more users take the advantage of services and resources offered by cloud networks, scalability also becomes an important issue, and this calls for careful resource monitoring, and management tools such as auto-scaling. Technically, auto-scaling can be seen as a proactive service that monitors the utilization levels of virtual resources across the infrastructure and scales the resources in use in response to estimates of future demand. However, resources are highly distributed and heterogeneous so future workload estimation is not always possible or sufficiently accurate. As a result, these methods are generally designed to fit the needs of a particular type of applications using a specific set of underlying measurement parameters.

Another approach to auto-scaling is to use pre-defined thresholds and real-time utilization measurements of virtual resources. Unfortunately, these methods are prone to false positives as the workload changes dynamically over time. In addition, they do not support horizontal scaling and are less capable in meeting the needs of specific application types. On the other hand, they do support vertical scaling, a useful component when combined with application-specific methods. A successful example of threshold based tools is Ceilometer, the open source telemetry component of OpenStack, that provides monitoring and auto-scaling services for heterogeneous virtual resources such as virtual computers (6). Ultimately, if thresholds are fixed, then and are defined by users which limits the functional scope of this component. OpenStack makes use of other measurement and analytics tools such as Cloud View (7) which are also limited in the same manner.

In this paper, we propose a vertical scaling method that improves the basic capabilities and the performance of a fixed threshold-based method with an adaptive controller that uses real-time utilization data to analyze and detect the appropriate scaling time. This adaptation makes our method less prone to false positives and allows it to be used in conjunction with any proactive horizontal scaling method to achieve better performance. The Smart Application on Virtual Infrastructure (SAVI) test bed (8) has a monitoring and measurement component, called Monarch (9). Monarch is logically centralized but physically distributed and supports monitoring and measurement of large infrastructures. It uses Ceilometer agents to monitor the resources in the compute layer to provide basic measurement and scaling features. Our assessment of our new method is done by combining it with the Monarch component in SAVI. In section II we continue with a description of our

978-3-901882-89-0 @2017 IFIP

proposed method and the performance evaluation is presented in section III. We discuss future work in section IV.

II. PROPOSED MODEL

Our method attempts to improve the performance delivered by fixed threshold methods, by introducing an adaptive vertical scaling feature. We define an algorithm to dynamically update the thresholds, that is up and down scaling thresholds, based on the real-time data utilization of the virtual resources. Therefore, the thresholds are no longer fixed and are dynamically being adjusted as the behavior of a particular virtual resource changes.

A. Dynamic Threshold Estimation

To support the dynamic updating feature, we use two separate methods for up and down scaling. Generally, the two methods monitor the virtual resources as separate tasks but share many basic features. In essence, our approach is more patient that the fixed threshold approach. When a threshold is exceeded, our method initiates an observation period to ascertain that the usage continues to exceed the threshold. Thus unnecessary scaling can be avoided, leading to improved resource usage. First, this increased efficiency translates into reduced cost. Second, a balance is created between performance and efficiency which is very important in providing a scalable application deployment environment. The general scope of our algorithm is shown in Fig. 1. As it can be observed, data from resources such as CPU and memory are being collected regularly. Collected data will then be sent to the data analysis component where the scaling decision will be made.

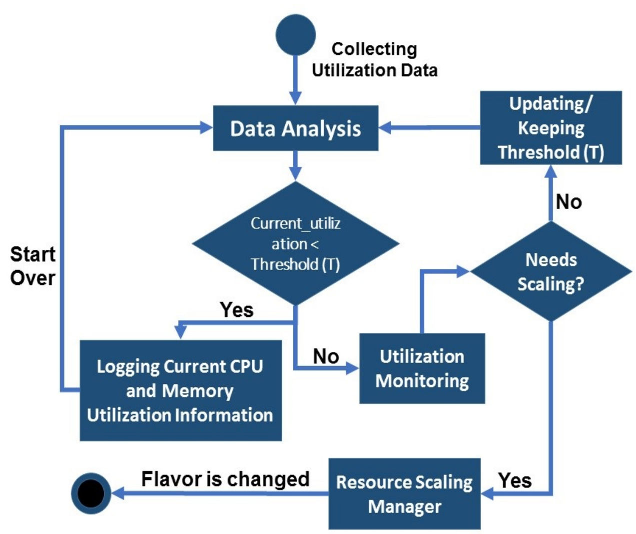

Fig. 1. Decision making process in the proposed Dynamic Threshold based Method

Our algorithm uses two parameters α and β. In up scaling, the parameter α is used to increase T (initial threshold) by α percent and the second parameter β indicates how many seconds our algorithm will monitor the ongoing condition if a threshold is met. In down scaling, α is used to decrease T (initial threshold). In up scaling, the algorithm starts by analyzing the CPU and memory utilization with a user-defined initial value for T. In general, if the threshold is met, β seconds are spent to monitor the ongoing situation. If after β seconds, the final value of the CPU or memory utilization remains above the threshold T, the value of threshold T is increased by α percent, so long as the increment in the value of T does not exceed the maximum possible utilization level or a hundred percent (Full Utilization). By design, when the algorithm observes a threshold exceeded, it hypothesizes that the situation is getting worst. It waits for β seconds to prove its hypothesis. If the condition persists after β seconds, the hypothesis is confirmed and the threshold (T) is updated. Otherwise, the hypothesis is rejected and the threshold is left unchanged. This scenario is continued, until the threshold can no longer be increased. At this point, up scaling will be triggered if still higher utilization values are observed. We need to select the initial value (T) and α so that the algorithm does not meet a hundred percent utilization threshold at the first iteration. The details of this algorithm are as follows.

Algorithm 1: UP Scaling Method

Result: UP Scaling Decision
initialization;
while *VM_State is Running* **do**
    if *VM_Current_Utilization* $>=$ *T* **then**
        if *New_Threshold* $<$ *(100 –* α*)%* **then**
            Monitor VM Utilization Level for β Seconds;
            if *VM_Current_Utilization* $>$ *T* **then**
                T = T + (Tα);
                Stop and Go to While;
            end
        else
            Execute UP Scaling Process;
        end
    end
end

Similarly, down scaling uses Tmin as the parameter that indicates the minimum possible tolerance level for which threshold (T) can be decreased. During the monitoring, if the threshold (T) is met, the analysis phase starts and continues to monitor the ongoing situation for a period of β seconds. If at the end of this period, the final captured utilization level remains below Tmin, it instantly triggers the down scaling signal. If not, and the utilization level is less than the current threshold (T), the algorithm temporarily decreases the current threshold by α percent and it continues to observe the situation for another β seconds. If the final utilization level is still less than the threshold value (T) but greater than Tmin, it commits the changes and decreases the threshold (T) by α percent. The details of this algorithm are below.

Algorithm 2: Down Scaling Method

```
Result: Down Scaling Decision
initialization;
while VM_State is Running do
    if VM_Current_Utilization <= T then
        if New_Threshold >= (Tmin)% then
            Monitor VM Utilization Level for β Seconds;
            if VM_Current_Utilization < T then
                Monitor VM Utilization Level for β
                Seconds;
                if VM_Current_Utilization < T then
                    T = T - (Tα);
                    Stop and Go to While;
                else
                    Decision = Ripeness Function();
                    /* Decision equals to 1
                       means to Downgrade the
                       Threshold                */
                    if Decision == 1 then
                        T = T - (Tα);
                        Stop and Go to While;
                    end
                end
            end
        else
            Execute Down Scaling Process;
        end
    end
end
```

We note that there is an important event that could happen during the second analysis time; the CPU or memory utilization exceeds the current threshold (T). This situation must be handled with care. The event could be a temporary burst which will be stabilized fast and thus has the potential to mislead the ongoing process. To avoid being affected by this issue, we introduce a ”Ripeness Function” (RF). When faced with this event, the RF triggers an additional analysis time to investigate the ongoing situation for another β seconds. As a result, the temporary value for the threshold will be set if the observed value at the end of the execution of Ripeness Function is less than threshold (T). Otherwise, the temporary threshold (T) will not be set. The RF is specifically designed to avoid false positives in down scaling. It should be noted that the third analysis phase that is triggered by RF is different from a normal analysis phase as it does not have another commitment phase to make a decision.

III. Performance Evaluation

To evaluate our algorithm, we installed Wordpress on a VM on SAVI with two CPU cores, 2GB of RAM and high bandwidth connectivity. We also created a test scenario to measure the CPU and memory utilization of our VM in which an increasing number of users attempt to connect to our VM and pull the data from our server, using Apache JMeter. We defined two test cases to compare the performance of a fixed threshold based method such as provided by Ceilometer and our proposed algorithm. In the first test case, we considered CPU utilization as a function of time to measure the performance of aforementioned methods in an up scaling scenario. We also defined the up scaling initial threshold (T) to be 70%. As can be seen in Fig. 2, on the left, the fixed threshold method executes the up scaling procedure to increase the number of available cores during the early moments of our test whereas our proposed method did not execute it at all. In a similar situation, we considered memory utilization as a function of time to assess the down scaling performance of the two methods. As shown in Fig. 2, on the right, the fixed threshold based algorithm triggered the down scaling procedure for our VM soon after it started to accept new connections. However, our proposed method did not triggered the down scaling procedure as it observed a higher utilization during its analysis phase (β).

In addition, note that the parameters α and β could be changed dynamically over time. However, we leave it open to users to make their decisions to either use fixed values or a mechanism to update them dynamically. Our suggestion to improve the performance of our algorithm and to take the most advantage of real-time data collection is to consider β as a function of the number of times the threshold (T) is being met. That means, the analysis time (β) can be shortened if the number of times the threshold (T) is met will be increased or in another word, the slope at which the threshold (T) is being met will be increased. In a similar way, parameter α can also be considered as a function of the rate at which the value of parameter (T) is being changed over time. Apparently, the higher rate will increase the chance of observing an appropriate up or down scaling faster.

IV. Conclusion and Future Work

As it has been discussed so far, our proposed method has several advantages in compare with basic features of fixed threshold based methods. First, it leverages the use of the real-time data collection to have a broader look on the ongoing situation. Besides, it considers the overall idle overhead of CPU and memory as a parameter to achieve an efficient vertical scaling objective. As aforementioned, using overall idle overhead is beneficiary due to the fact that it can be translated to the actual cost for both the infrastructure owners and their users (accountability) and also provides balance between performance and efficiency in use. In addition, enhancing the basic features introduced by fixed threshold based monitoring and measurement tools such as Ceilometer can also help to improve the features of advanced monitoring tools such as Monarch. Besides, vertical scaling methods can be always seen as an important feature of a cloud computing platform and can also be used in conjunction with any other horizontal scaling methods to provide full support of scalability as an important parameter for the survivability of the whole infrastructure.

Our method can be improved in several ways. First, as we discussed, parameters α and β can be dynamically adjusted in regards to the frequency at which the threshold (T) is met.

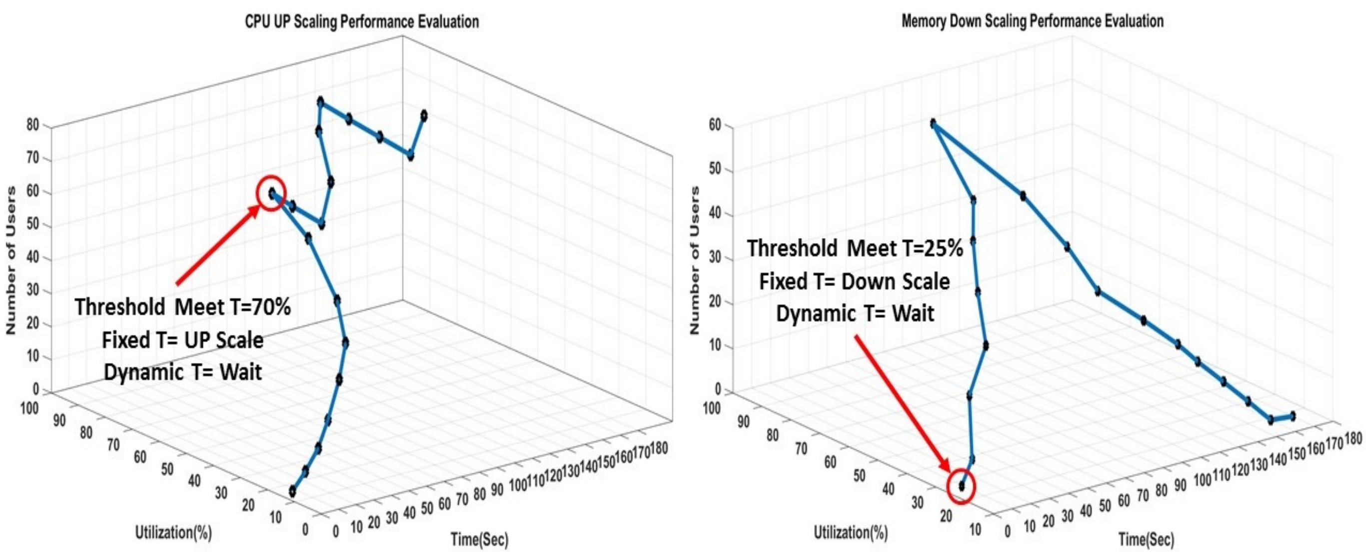

Fig. 2. Performance Evaluation of Dynamic versus Fixed Threshold Method in an up scaling scenario (Left) and a down scaling scenario (Right)

However, it is an open part of our work and more sophisticated functions can be considered for that reason. Besides, one may prefer to add more parameters to our algorithm to improve the accuracy and correctness delivered by it. For instance, the possibility of returning to the previous configuration (flavor) after any up or down scaling event can be fed into our model as an indicator of the accuracy of the previously made decision. In this way, our method can be able to even learn the appropriate values of α and β from their effects in the future status of the corresponding VM. Quality of Service (QoS) parameters can also be considered to reflect the applications' requirements. Typically, QoS parameters are expressed in terms of Service Level Agreements (SLAs). Each time the scaling decision will be made, careful assessment of SLAs has to be done to avoid SLA violations. However, it is a valid decision to sometimes let SLA violations to occur to avoid insatiability. QoS parameters and in particular SLAs can also help to estimate more accurate values for parameters α and β as in this way, the number of times scaling happens is affected more by application requirements or in another word, users' expressions of their needs.

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. L. Massie, B. N. Chun, and D. E. Culler, "The ganglia distributed monitoring system: design, implementation, and experience," *Parallel Computing*, vol. 30, no. 7, pp. 817–840, 2004.

[3] A. Vladishev, "Zabbix. an enterprise-class open source distributed monitoring solution," *Juli-2010.[Online]. Available: http://www. zabbix. com/.[Accessed: 21-Juli-2010]*, 2007.

[4] C. Yu, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang, and H. V. Madhyastha, "Flowsense: Monitoring network utilization with zero measurement cost," in *International Conference on Passive and Active Network Measurement*. Springer, 2013, pp. 31–41.

[5] S. R. Chowdhury, M. F. Bari, R. Ahmed, and R. Boutaba, "Payless: A low cost network monitoring framework for software defined networks," in *Network Operations and Management Symposium (NOMS), 2014 IEEE*. IEEE, 2014, pp. 1–9.

[6] J. Lin, "Monarch: Scalable monitoring and analytics for visibility and insights in virtualized heterogeneous cloud infrastructure," Ph.D. dissertation, University of Toronto, 2015.

[7] P. Sharma, S. Chatterjee, and D. Sharma, "Cloudview: Enabling tenants to monitor and control their cloud instantiations," in *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*. IEEE, 2013, pp. 443–449.

[8] J.-M. Kang, H. Bannazadeh, and A. Leon-Garcia, "Savi testbed: Control and management of converged virtual ict resources," in *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*. IEEE, 2013, pp. 664–667.

[9] J. Lin, R. Ravichandiran, H. Bannazadeh, and A. Leon-Garcia, "Monitoring and measurement in software-defined infrastructure," in *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*. IEEE, 2015, pp. 742–745.