Auto Scaling Strategy for Amazon Web Services in Cloud Computing

Wen-Hwa Liao
Department of Information Management
Tatung University
Taipei, Taiwan
E-mail: whliao@ttu.edu.tw

Ssu-Chi Kuai and Yu-Ren Leau
Department of Information Management
Tatung University
Taipei, Taiwan

Abstract—**Auto scaling mechanisms have become a typical paradigm in cloud computing environments. Such mechanisms can increase or minimize the number of virtual machines according to user demands, consequently achieving pay-per-use objectives. However, auto scaling mechanisms provided by infrastructure-as-a-service providers must strictly follow user-defined thresholds; the drawback of such mechanisms is that they cannot respond to real-time Internet traffic loads by following user-defined thresholds. Therefore, we propose a dynamic threshold adjustment strategy that can expedite the creation of virtual machines according to workload demands. The proposed strategy can reduce the web application response time and error rate when the system is under a heavy workload. In addition, it can expedite the release of virtual machines to reduce virtual machine running time when the system is under a light workload. According to our experimental results, we found that CPU-intensive web applications require an excellent threshold control strategy. Therefore, the proposed strategy can satisfy this requirement by effectively reducing the response time of applications, virtual machine running time, and error rate.**

Keywords—auto scaling; cloud computing; dynamic threshold

I. Introduction

In recent years, cloud computing has emerged as a tool for solving problems involving specific demands for computing resources [1]. With the rapid development of virtualization technology, computing resources can be used at any time and location with lower cost and stronger performance [2]. As with the application of electricity, water and telephone services, and evaluation models, cloud computing has become popularized and has served as the paradigm of new computing models, such as Amazon Web Services (AWS) [3]. Nevertheless, the fluctuating demand for computing resources is a typical challenge for enterprises during the application of cloud computing. Consequently, how to dynamically increase or reduce resources has become a crucial research topic and the key to effective cloud computing. In general, infrastructure-as-a-service providers offer a user-defined threshold for triggering the automatic increase or minimization of virtual machine resources. However, using a fixed threshold for triggering virtual machine resources is not an optimal approach because different web applications may share a physical computing resource or virtual machine [4][5][6][7][8][9]. In addition, a system's threshold adjustment scheme should be determined according to the workload of a specific application. Despite the complicated topological environment of current web applications, thresholds should be adjusted according to the demand for resources at a specific time. In other words, when the demand for resources is higher, the threshold for increasing the number of virtual machines for satisfying such a demand should be reduced (and vice versa); doing so ensures a feasible dynamic threshold setting process. Therefore, in this study, we proposed a dynamic threshold adjustment strategy and implemented it on the AWS platform to improve this platform's inherent fixed threshold mechanism that is used for creating and terminating virtual machines. The proposed strategy thus provides an adaptive mechanism for increasing or minimizing virtual machine resources for users.

The rest of this paper is organized as follows. Section II reviews the related work. Section III describes our auto scaling strategy for AWS. Simulation results are described in Section IV, and Section V concludes the paper.

II. Related Work

The auto scaling mechanism in AWS enables users to define a fixed threshold that serves as the condition for triggering virtual machine scaling processes. However, numerous studies have indicated that this auto scaling mechanism is not suitable for environments involving dynamic and unpredictable network loads [4][5][6][7][8][9].

In the mechanism in [10], when a workload within a specific time causes the failure of the system response time to satisfy quality of service (QoS) targets, the request of each virtual machine is rejected. Each virtual machine calculates the system response time to compute the average response time. When the average response time cannot satisfy the QoS targets, the number of virtual machines is recalculated. The number of virtual machines is then increased; the proportion of increased virtual machines is equivalent to half the number of current virtual machines. The increased number of virtual machines does not exceed the default value of the maximum number of virtual machines. Regarding the mechanism for releasing the virtual machines, the authors adopted a measure similar to the fixed threshold mechanism. When the resource use is lower than a default upper threshold, the virtual machines are released; the number of released virtual machines is equal to half the minimum number of virtual machines plus the maximum default number of virtual machines. The number of released virtual machines cannot be less than the minimum default number of virtual machines. However, this method is not sufficiently flexible, and excessively terminating virtual machines may engender the problem of overtermination or overcreation. In addition, frequently terminating or creating virtual machines may result in unnecessary cost and the inability to quickly respond to the demands of web applications.

In [5], the authors indicated that the frequent creation or termination of virtual machines results in unnecessary cost. Therefore, such a situation should be avoided in the design of auto scaling mechanisms. They proposed a measure that can prevent excessively frequent creation or termination of virtual machines when operating under bursty workload demands; this measure requires a user-defined upper threshold and lower threshold, and it sets a range to control the threshold-triggering mechanism. Despite the effectiveness of the preceding two measures, a dynamic threshold adjustment scheme has yet to be achieved. Moreover, trigger thresholds cannot be dynamically adjusted to respond to the conditions of virtual machines. Although the standard hysteresis mechanism can control the scaling process in accordance with the usage state, it delays the speed at which the number of the virtual machines is increased, and even fails to respond to bursty workload demands of web applications [7].

In [11], the authors focused on a single indicator, such as a CPU, to simplify the problem of virtual machine allocation. When the value of a CPU that is monitoring virtual machines exceeds the upper threshold for a specific period, the number of virtual machines is increased for balancing the workload to satisfy the increased demand of the virtual machines. When the CPU value becomes lower than the lower threshold for a predefined time, the number of virtual machines is reduced for cost saving. However, because of the dynamic behaviors of users when they utilize web applications, different resources must generally be expanded. For example, consider a situation entailing the allocation of resources to an application layer; in this situation, the authors used a strategy tree to propose three strategies as the criteria of dynamic resource allocation: sensitive, tolerant, and aggressive strategies. However, the three strategies cannot adequately resolve bursty workload demands of web applications.

III. Auto Scaling Strategy for AWS

We implemented an auto scaling strategy for the AWS platform to provide virtual machine resources that serve as computing resources for web applications. Resource usage and CPU utilization values are monitored using the CloudWatch service of AWS. These monitored values are transferred to a dynamic threshold controller for analysis. The controller transmits a timely policy and threshold to the virtual machines for providing computing resources. Fig. 1 displays the system architecture.

A. *Dynamic Threshold Adjustment Mechanism*

In the auto scaling of resources in cloud environments, it is highly imperative to select a suitable upper threshold to increase the number of virtual machines and a lower threshold to reduce the number of virtual machines. This is because an excessively high threshold reduces the QoS of applications, and an extremely low threshold results in unnecessary cost. However, related studies have not provided an optimal definition of upper and lower CPU utilization thresholds. Additionally, the workload of web applications is dynamic. Specifically, users cannot comprehend the timing of peak. Hence, this study focused on a dynamic threshold adjustment mechanism.

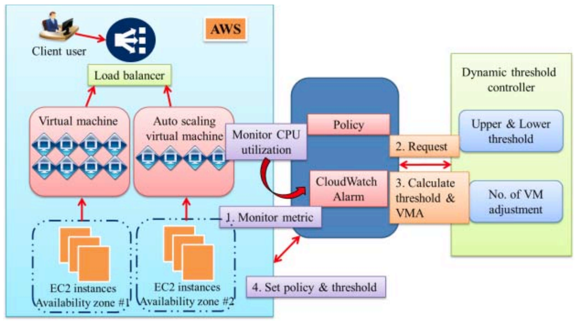
AWS
Client user
Load balancer
Virtual machine
Auto scaling virtual machine
Monitor CPU utilization
Policy
CloudWatch Alarm
1. Monitor metric
2. Request
3. Calculate threshold & VMA
4. Set policy & threshold
Dynamic threshold controller
Upper & Lower threshold
No. of VM adjustment
EC2 instances Availability zone #1
EC2 instances Availability zone #2

Fig. 1. System architecture.

The workload of web applications is typically dynamic, and predicting the work schedule is difficult; therefore, setting dynamically adjusted thresholds is highly appropriate. As described in [8], when virtual machines use 100% of CPU resources, the resource consumption level of an application service per unit is raised. The authors in [12] reported that specific operating systems such as Linux do not ensure that high CPU load contributes to high performance. They also indicated that the optimal upper threshold in diverse situations should be set to approximately 50%–75%. The experimental results in another study confirmed that when virtual machines are used in a CPU-intensive application on an Apache server, the upper threshold should be set to nearly 50%–75% [9]. The authors in [18] presented a resource optimization scheme for dispatching resources to virtual machines in a cloud data center, and the experimental results revealed that the default upper threshold of CPU utilization was 70%. According to the preceding studies, setting the upper threshold of CPU resource utilization to approximately 50%–75% is favorable for divergent web application. Because we did not consider the types of web application, we set the upper threshold of CPU utilization to nearly 50%–75%.

Regarding the method of triggering the upper threshold adjustment mechanism, we claim that when more than 50% of the virtual machines in a virtual machine set are expanded, half of the nodes are under a heavy workload. To prevent web applications from being delayed to enable them to respond to overloading faster, when the upper threshold is set in the range 50%–75%, 0.5% of the upper threshold decreases proportionally with the expansion process per unit; in other words, $(50\%-75\%)/(100-50) = (-0.5\%)$. Consequently, in the virtual machine set in this study, when 1% of the machines are expanded, the upper threshold of the CPU utilization decreases (by 0.5%) linearly with the CPU threshold ranging from 50% to 75%.

When more than 50% of the virtual machine collections do not expand, the workload is alleviated, and the upper threshold must be raised moderately to avoid a highly sensitive threshold policy, resulting in lower cost. When the threshold is set to nearly 50%–75%, 0.5% of the upper threshold increases proportionally with the expansion of virtual machines per unit; in other words, (75%−50%)/(100−50) = 0.5%. When 1% of the virtual machines in the virtual machine set are not expanded, the CPU upper threshold increases by 0.5% linearly, and the CPU threshold ranges from the strictest 50% to the loosest 75%. The upper threshold is not raised until no virtual machines are expanded for a specific period.

In a traditional cloud data center, virtual machines with a CPU utilization ranging from 0% to 12% are in a free status, those with a CPU utilization ranging from 30% to 67% are in a general status, and those with a CPU utilization exceeding 68% are in a busy status. In addition, progressively failing to fully use servers results in lower cost, which has become a critical problem in information technology [13]. In [13] and [14], the authors have not analyzed or defined the percentage of servers that do not result in unnecessary cost; however, they have generally maintained that when the CPU utilization is less than 20%, relative measures, such as changing the work schedule or triggering the auto scaling mechanism to terminate the virtual machine, should be adopted to prevent the loss of cost benefits. Therefore, in the current study, the dynamic lower threshold of CPU utilization was designed in the range 5%–30%.

The method we implemented for triggering the lower threshold adjustment mechanism is described as follows. When more than 50% of the virtual machines in a virtual machine set are terminated, we consider that the traffic of web applications is extremely low and that numerous machines are rendered redundant, which may result in unnecessary cost. Consequently, the lower threshold is increased, and this relaxed threshold policy may thus achieve a faster termination of the virtual machines. Moreover, because the threshold ranges from 5% to 30%, the release proportion per unit increases by 0.5%; that is, (30%−5%)/(100−50) = 0.5%. Therefore, when 1% of the machines in the virtual machine set are released, the CPU lower threshold is increased linearly at a rate of 0.5%, and it ranges from 5% (strictest) to 30% (loosest).

When more than 50% of the virtual machines in the virtual machine set are not released, to prevent unnecessary cost from being induced by the frequent creation or termination, the lower threshold is reduced to minimize the probability of terminating the virtual machines. In addition, the lower threshold is set to the range 5%–30%. Therefore, the lower threshold is reduced by 0.5% when the proportion of nonreleased virtual machines increases by 1%; that is, (5%−30%)/(100−50) = (−0.5%). Specifically, when the proportion of the nonreleased machines increases by 1%, the lower threshold of CPU utilization linearly drops by 0.5%. The authors in [14] reported that excessively frequent creation and termination of virtual machines causes the loss of cost benefits.

B. Dynamic Threshold Adjustment Strategy

To promptly respond to a considerably high bursty workload demand of web applications, the method of dynamically minimizing the upper threshold is applied. In this method, the upper threshold decreases linearly by 0.5% when the proportion of virtual machines increases by 1%. Moreover, the condition of creating the virtual machines is relaxed in this method. The purpose of raising the upper threshold is to minimize the workload of web applications and avoid the unnecessary creation of virtual machines, thus preventing unnecessary cost. The formulas for calculating the upper threshold for a specific time are expressed subsequently, where U^- represents the reduction of the upper threshold, U^+ represents the increase of the upper threshold, and U_P is the interval between the current time and the previous time at which the virtual machines are expanded. The calculation method is $|(N_{VM}(t) - N_{VM}(t-1)) / N_{VM}(t-1)|$, where $N_{VM}(t)$ is the number of the virtual machines at the current moment t. Furthermore, U_B is the standard rate at which the proportion of virtual machines is changed in the expansion process, and it is set to 50%. The upper limit of the range of the upper threshold U_{Max} is 75%. The lower limit of the range of the upper threshold U_{Min} is 50%. The number of expanded virtual machines is U_{VMA}. The term $VMSet$ represents the set of all virtual machines. The method of reducing the upper threshold is expressed in (1), and that of raising the upper threshold is expressed in (2).

$$U^+ = U_{Min} + (1 - (U_P - U_B)) \times \frac{U_{Max} - U_{Min}}{U_B}. \quad (1)$$

$$U^- = U_{Max} + (U_P - U_B) \times \frac{U_{Min} - U_{Max}}{U_B} \quad (2)$$

The purpose of dynamically raising the lower threshold is to avoid the excessively frequent creation of web application virtual machines that are in an idle status to save unnecessary cost. The lower threshold increases linearly by 0.5% when the number of released virtual machines increases by 1%. The objective of dynamically minimizing the lower threshold is to reduce it to a reasonable value when the machines are no longer idle. Similarly, for the linear method, when the number of unreleased virtual machines increases by 1%, the lower threshold is reduced by 0.5% proportionally. The formula for calculating the upper threshold for a specific time is expressed subsequently, where L^+ is the lower threshold that is raised at the moment, L^- is the lower threshold reduced at the moment, and L_P is the difference between the current time and the previous time at which the virtual machines are released. The computation method is $|(N_{VM}(t) - N_{VM}(t-1)) / N_{VM}(t-1)|$. The upper limit of the threshold range L_{Max} is 30%. Moreover, L_B is the standard rate at which the proportion of virtual machines is changed in the release process, and it is set to 50%. The lower limit of the threshold range L_{Min} is 5%. The number of released virtual machines is L_{VMA}. The method of calculating the lower threshold for increasing is expressed in (3), and that of calculating the lower threshold for reduction is expressed in (4).

$$L^- = L_{Max} + (1 - (L_P - L_B)) \times \frac{L_{Min} - L_{Max}}{L_B}. \quad (3)$$

$$L^+ = L_{Min} + (L_P - L_B) \times \frac{L_{Max} - L_{Min}}{L_B}. \quad (4)$$

Fig. 2 shows the algorithm of auto scaling strategy. When the upper threshold falls between U_{Min} and U_x, it is considered an extreme situation. Therefore, the general policy is changed to the sensitive policy, and the number of increased virtual machines is changed to 2 in response to the high amount of workload. In addition, U_x is 52.5%, which is calculated according to the expression $U_{Min} + (U_{Max} - U_{Min}) \times U_M$. In this paper, U_M is set to 10%. By contrast, when the lower threshold falls between L_x and L_{Max}, the workload set in the virtual machine is assumed to be extremely low; therefore, the number of released virtual machines is changed to 2 for preventing unnecessary cost. We can set L_x to 27.5%, and it is calculated as follows: $L_{Max} - (L_{Max} - L_{Min}) \times L_M$. Moreover, L_M is set to 10%.

Algorithm 1: Auto scaling strategy

```
if ($U_p == 0$) then
    $U_{Max} = U^+$;    //By Eq.(1)
    break;
else
    $U_{Max} = U^-$;    //By Eq.(2)
end
if ($U \in [U_{Min}, U_x]$) then    //U is utility of resource
    $U_{VMA} = 2$;
else
    $U_{VMA} = 1$;
end
if ($L_p == 0$) then
    $L_{Max} = L^-$;    //By Eq.(3)
    break;
else
    $L_{Max} = L^+$;    //By Eq.(4)
end
if ($L \in [L_x, L_{Max}]$) then    //L is utility of resource
    $U_{VMA} = -2$;
else
    $U_{VMA} = -1$;
end
```

Fig. 2. Algorithm of auto scaling strategy.

IV. Simulation Results

To demonstrate the effectiveness of the proposed dynamic threshold adjustment algorithm (Dy.), we compared it with a traditional fixed threshold method (Fix.) [7] and an adaptive virtual machine provisioning algorithm (Ad.) [10]. A web application was implemented on the AWS platform, and a backend MySQL database was constructed. We used Amazon Elastic Compute Cloud (EC2) virtual machines, which involved an elastic load balancing mechanism, and the dynamic threshold adjustment strategy to compare the response times of applications and virtual machine running times; these comparisons were conducted to calculate the cost and error rates of servers for evaluating the probability of such servers responding to workload demands. The QoS target was set as an indicative type of web application involving the metrics of response time and error rate. In a text stream web application, the response time indicator is 2–4 s, and the error rate is less than 2%. In the video stream web application, the response time is 2–5 s, and the error rate is less than 4%. Apache JMeter is an open source software program designed for extensively testing the performance of applications [15]. Therefore, we used Apache JMeter to measure the response time and error rate. The experimental environment was on the AWS platform, and EC2 t1.micro virtual machines were adopted. The specification of the virtual machines is provided in Table I, and this machine is run on Linux CentOS.

TABLE I. EC2 Virtual Machine Parameters

System parameters	Value
Instance type	t1.micro
CPU	1vCPU
Memory	613 MB
Storage	0.615 GB
Price	\$0.020/h

A. Experimental Workload

Fig. 3 illustrates the dynamic workload derived by simulating the possible workload in the application environment. The vertical axis represents the number of requests per minute.

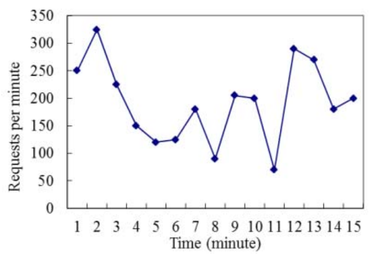

Fig. 3. Experimental workload.

B. Average Response Time

The response time of web applications directly influences a user's usage intention and signifies the efficacy of such applications. Consequently, in this experiment, we compared the response times of the fixed threshold, dynamic threshold, and adaptive virtual machine provisioning algorithms.

In the experiment, Apache JMeter was run 10 times for 15 min to calculate the average response time, virtual machine running time, and error rate. For the text stream web application, the response time for a specific demand involves the total elapsed time from when the application calls the backend database after transmitting the demand to when it responds to the text data demand. For the video stream web application, the response time for a specific demand involves the total elapsed time from when the application calls the backend database after transmitting the demand to when it responds to the demand of video data for coding. The variation of the response time in the entire operation was logged using the Apache JMeter.

Fig. 4 displays the average response time of the text stream web application, and Fig. 5 illustrates that of the video stream web application. According to the experimental results, a lower upper threshold did not always shorten the response time of the text stream web application. It instead minimized each CPU demand of the text stream web application. Therefore, threshold adjustment does not significantly influence response time. By contrast, for the video stream web application, an upper threshold adjustment exerted a more significant influence on the response time because each CPU demand of the text stream web application was higher. Hence, CPU-intensive web applications must have strong scaling mechanisms for adjusting thresholds.

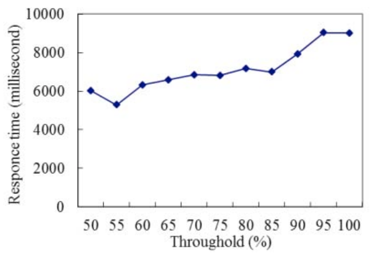

Fig. 4. Response time of text stream applications with fixed threshold.

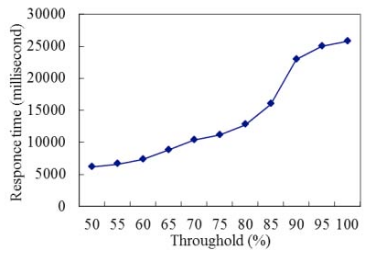

Fig. 5. Response time of video stream applications with fixed threshold.

The delay time that a user can accept is approximately 0–5000 ms [16]. The response time of the proposed dynamic threshold adjustment strategy is within this range. The experimental results provided in Figs. 6 and 7 indicate that the proposed strategy can more effectively reduce the response time of both the text stream and video stream applications, achieve higher web efficacy, and improve the response time of CPU-intensive web applications compared with the other algorithms.

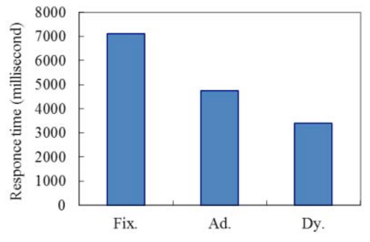

Fig. 6. Comparison of response time of text stream applications.

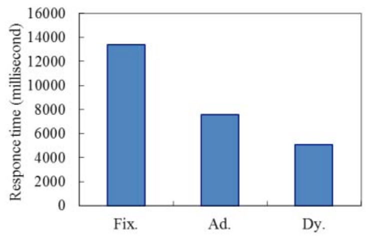

Fig. 7. Comparison of response time of video stream applications.

C. Running Time of the Virtual Machines

In the experiment, we also compared the cost of using various traditional fixed threshold methods and the adaptive virtual machine provisioning algorithm. Because the computational time of rented virtual machines in cloud computing environments is actually equivalent to the cost, we compared the total running times of all virtual machines in the entire testing process (Figs. 8 and 9).

According to the experimental results illustrated in Fig. 8, a lower threshold resulted in a longer virtual machine running time and consequently a higher cost. However, an extremely high and fixed lower threshold caused unnecessary termination of the virtual machines, further resulting in the problem of a longer response time. In the video stream web application, each demand consumes a high amount of CPU resources; therefore, at a lower threshold, the termination of the virtual machines was less frequent, and the machines were terminated at a later time (Fig. 9). When the lower threshold was less than 20%, the running time of the virtual machines was poor. For the adaptive virtual machine provisioning algorithm, half the virtual machines were created. Therefore, although the response times of the virtual machines were sufficient, this algorithm cannot effectively reduce the response time of web applications and save the running cost simultaneously.

The dynamic threshold adjustment strategy can save considerable cost because it raises the lower threshold and quickly reduces the number of virtual machines in a virtual machine set from two machines to one machine under light workloads.

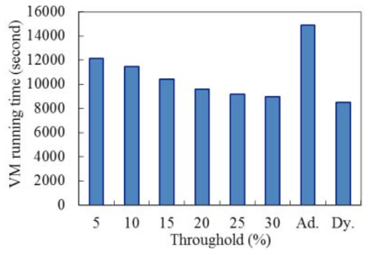

Fig. 8. Running time of text stream virtual machines.

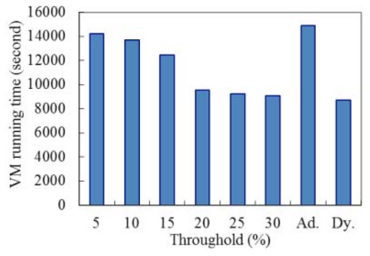

Fig. 9. Running time of video stream virtual machine.

D. Error Rate

The error rate of web applications refers to the probability of a server rejecting a demand. The occurrence of an error may indicate the failure of virtual machines to respond. In general, such a failure is caused when a server rejects considerably high bursty workload demands because it cannot urgently process them. The error rate is also one of the parameters used to evaluate web applications (Figs. 10 and 11).

The experimental results displayed in Fig. 10 reveal that the error rate of the text stream web application begins to rise when the fixed upper threshold is 85%. The experimental results provided in Fig. 11 indicate that the error rate of the video stream web application begins to rise sharply when the fixed upper threshold is 90%. This may be attributed to the failure of the server to respond when an extremely heavy workload is imposed on the virtual machines without increasing the number of virtual machines. Regarding the proposed dynamic threshold adjustment strategy, the server error rate for the text stream web application was approximately 1.21%; this low error rate is attributable to the ability of the proposed strategy to quickly create virtual machines. For the traditional fixed threshold method, a higher error rate was observed when the workload was increased rapidly. When the dynamic threshold adjustment strategy was applied to the video stream web application, the server error rate was nearly 2.38%. Because the adaptive virtual machine provisioning algorithm creates half the virtual machines, in the text stream web application, the error rate was lower (i.e., 1.44%), whereas that for audio–video web applications was 2.32%.

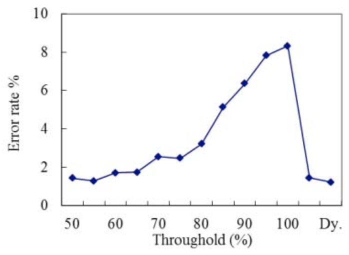

Fig. 10. Error rate of text stream web application.

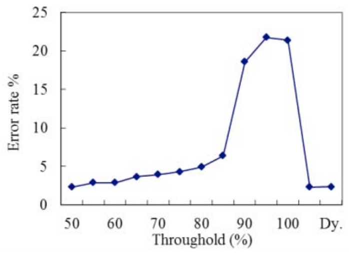

Fig. 11. Error rate of video stream web application.

V. Conclusion

Auto scaling mechanisms play a highly crucial role in cloud computing environments. Such mechanisms are typically used to respond to bursty workload demands or terminate unnecessary virtual machines to save operating cost. However, the traditional fixed threshold strategy or methods that entail creating virtual machines according to demand fail to truly respond to bursty workload demands or result in increased cost. Therefore, this paper proposes a dynamic threshold adjustment strategy that reduces the threshold of virtual machines at peak workload demands and raises the threshold when the virtual machines are idle, thus further saving the cost. According to our experimental results, the dynamic threshold adjustment strategy can effectively reduce the response time of applications, error rate, and virtual machine running time. The results also revealed that CPU-intensive web applications must adopt strong threshold control strategies. The proposed dynamic threshold adjustment strategy significantly influences web-based CPU-intensive applications that consume a high amount of CPU resources per request.

Acknowledgments

This work was supported in part by the Ministry of Science and Technology, Taiwan, under grant MOST104-2221-E-036-005, and Tatung University, under grants B104-N01-058 and B104-N02-059.

References

[1] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, “Cloud Computing and Emerging It Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility,” Future Generation Computer Systems, Vol. 25, No. 6, pp. 599-616, 2009.

[2] Y. F. Li, W. Li, and C. F. Jiang, “A Survey of Virtual Machine System: Current Technology and Future Trends,” IEEE International Symposium on Electronic Commerce and Security, 2010.

[3] Amazon Web Service, Amazon Inc., http://aws.amazon.com/

[4] A. Beloglazov and R. Buyya, “Adaptive Threshold-based Approach for Energy-efficient Consolidation of Virtual Machines in Cloud Data Centers,” ACM International Workshop on Middleware for Grids, Clouds and e-Science, 2010.

[5] A. Beloglazov and R. Buyya,“Optimal Online Deterministic Algorithms and Adaptive Heuristics for Energy and Performance Efficient Dynamic Consolidation of Virtual Machines in Cloud Data Centers,” Concurrency and Computation: Practice & Experience, Vol. 24, No. 13, pp. 1397-1420, 2012.

[6] T. C. Chieu, A. Mohindra, and A. A. Karve, “Scalability and Performance of Web Applications in a Compute Cloud,” IEEE International Conference on eBusiness Engineering, 2011.

[7] M. Z. Hasan, E. Magana, A. Clemm, L. Tucker, and S. L. D. Gudreddi, “Integrated and Autonomic Cloud Resource Scaling,” IEEE Network Operations and Management Symposium , 2012.

[8] S. Srikantaiah, A. Kansal, and F. Zhao, “Energy Aware Consolidation for Cloud Computing,” USENIX HotPower: Conference on Power Aware Computing and Systems, 2008.

[9] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif, “Black-box and Gray-box Strategies for Virtual Machine Migration,” USENIX Symposium on Networked Systems Design & Implementation, 2007.

[10] R. N. Calheiros, R. Ranjany, and R. Buyya, “Virtual Machine Provisioning Based on Analytical Performance and QoS in Cloud Computing Environments,” IEEE International Conference on Parallel Processing (ICPP), 2011.

[11] B. Simmons, H. Ghanbari, M. Litoiu, and G. Iszlai, “Managing a SaaS Application in the Cloud Using PaaS Policy Sets and a Strategy-tree,” IEEE International Conference on Network and Service Management, 2011.

[12] A. Murtazaev and S. Oh, “Sercon: Server Consolidation Algorithm Using Live Migration of Virtual Machines for Green Computing,” IETE Technical Review, Vol. 28, No. 3, pp. 212-231, 2011.

[13] W. Vogels, “Beyond Server Consolidation”, ACM Queue, Vol. 6, No. 1, pp. 20-26, 2008.

[14] A. Ashraf, B. Byholm, J. Lehtinen, and I. Porres, “Feedback Control Algorithms to Deploy and Scale Multiple Web Applications Per Virtual Machine,” IEEE Euromicro Conference on Software Engineering and Advanced Applications, 2012.

[15] Apache Jmeter, Apache Software Foundation., http://jmeter.apache.org/

[16] Y. Song, H. Wang, Y. Li, B. Feng, and Y. Sun, “Multi-tiered on Demand Resource Scheduling for VM-based Data Center,” IEEE/ACM International Symposium on Cluster Computing and the Grid, 2009.