Auto Scaling Virtual Machines for Web Applications with Queueing Theory

Gaopan Huang[1], Songyun Wang[23], Mingming Zhang[1], Yefei Li[2], Zhuzhong Qian[4], Yuan Chen[4], and Sheng Zhang[4]
[1]State Grid Jiang Su Electric Power Company Information & Telecommunication Branch, P.R. China
[2]Jiang Su Frontier Electric Technology CO.LTD, P.R. China
[3]Nanjing University of Aeronautics and Astronautics, P.R. China
[4]State Key Lab. for Novel Software Technology, Nanjing University, P.R. China
{13645153778, zhangmms, imliyf}@163.com, {qzz@, chenyuan@dislab., sheng@}nju.edu.cn

***Abstract*—With the rapid development of cloud computing in recent years, more and more individuals and corporations use cloud computing platform to deploy their web applications, which can significantly minimize their deployment costs. However, it is observed that the number of accesses to some web application often fluctuates over time, resulting in the so-called peak-valley phenomenon: the amount of reserved resources is often proportional to the peak need 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. To solve this problem, we establish a queuing model M/M/C, which represents infinite source and multi-service window. Based on this queueing model, we can accurately predict the arrival time of each customer, which enables us to calculate the minimum amount of resources that meet the resource needs. Then, we use heuristic algorithms and dynamic programming method to design a Virtual Machine (VM) auto-scaling strategies, including horizontal scaling and vertical scaling. With the proposed model and scaling algorithms, we can make web applications not only meet customer needs, but also use the least amount of resources, improving the resource utilization and minimizing deployment costs. With extensive experiments, we show the proposed model and scaling algorithms can greatly improve resource utilization without sacrificing web application performance.**

***Index Terms*—Cloud-computing, auto-scale, web application**

I. Introduction

Cloud computing is a kind of Internet-based computing that provides shared processing resources and data to computers and other devices on demand. It is a model for enabling ubiquitous, on-demand access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications and services), which can be rapidly provisioned and released with minimal management effort. There are lots of cloud computing platforms, such as Amazon EC2, Windows Azure, and some open source platforms, such as Openstack and Cloudstack. In these platforms, a physical machine (PM) is usually divided into several virtual machines (VMs), which can be managed more conveniently. A VM can easily be launched, resized, or released. With the help of virtualization technology, we can dynamically deploy web applications in cloud environments.

As we know, the demands of web applications usually fluctuate [1–3]. When we deploy some web the application in a cloud, we often reserve physical resources that are proportional to the peak demand of an application. However, reservation for peak demand may waste physical resources, since most of the time the amount of required resources is far below the peak demand. Moreover, it would also increase the deployment cost for web application owner.

In fact, we can dynamically change the resource demand of the web application according to its history demands, and maintain the most suitable resources of the web application. In this paper, we establish a queuing model M/M/C, which represents infinite source and multi-service window. Based on this queueing model, we can accurately predict the arrival time of each customer, which enables us to calculate the minimum amount of resources that meet the resource needs. Then, we use heuristic algorithms and dynamic programming method to design a Virtual Machine (VM) auto-scaling strategies, including horizontal scaling and vertical scaling. With the proposed model and scaling algorithms, we can make web applications not only meet customer needs, but also use the least amount of resources, improving the resource utilization and minimizing deployment costs. The contributions of this paper are three-fold:

- We present the virtual machine auto-scaling problem and give the formal description based on queueing theory.
- We propose algorithms for calculating the optimal resource demands and auto scaling VMs.
- With extensive experiments, we show the proposed model and scaling algorithms can greatly improve resource utilization without sacrificing web application performance.

The rest of the paper is organized as follows. We discuss related work in Section II. We introduce the problem in Section III. We present our solution in Section IV. Before we conclude the paper in Section VII, we evaluate our design in Section VI.

II. Related Work

In general, there are four kinds of methods to solve dynamic resource deployment problem: threshold-based policies, reinforcement Learning, control theory, and queuing theory.

Threshold-based policies are very popular among cloud providers such as Amazon EC2, and third-party tools like Right Scale. The simplicity and intuitive nature of these

978-1-5090-5521-0/16/$31.00 ©2016 IEEE

policies make them very appealing to users. For example, if we monitor that the average CPU load of servers larger than 70% for 5 minutes, we add some servers; while the average CPU load of servers smaller than 30% for 5 minutes, we remove some servers. However, setting thresholds is a hard task, deciding the performance of the threshold-based policies.

Reinforcement learning (RL) is a type of automatic decision-making approach that can be applied for auto-scaling. It captures the performance model of a target application and its policy without any a priori knowledge [4–6]. Regarding the reward function, it usually takes into account both the cost of the resources and the cost derived from SLO violations. However, the process of reinforcement learning is too slow, this technique adapts only to slowly changing conditions.

Control theory has been applied to automate the management of web server systems, storage systems, data centers/server clusters, and other systems, and it also shows interesting results on cloud computing platforms [7–9]. Control systems are mainly reactive, but there are also some proactive approximations such as model predictive control, or even combining a control system with a predictive model. However, setting the gain parameters can be a difficult task, and it may cause assignment oscillations.

Queuing theory has been extensively used to model Internet applications and traditional servers, in order to estimate performance metrics such as the queue length or the average waiting time for requests [2, 3, 10, 11]. For example, the resource demand of a VM is modeled by a two-state Markov chain in [3], and a stationary distribution-based algorithm is proposed for scaling VMs. In the subsequent sections, we will describe the main characteristics of a queuing model and how they can be applied to scalable scenarios.

III. Problem Formulation

A. Preliminaries

1) Poisson distribution: Poisson distribution is a discrete probability distribution which expresses the probability of a given number of events occurring in a fixed interval of time. If random variable X follows Poisson distribution with parameter λ, the probability that X happens k times in unit time is

$$P(X = k) = \frac{\lambda^k}{k!} e^{-\lambda},$$

And the expectation of the number of times that X happens in unit time is

$$E(X) = \lambda.$$

2) Queueing theory: Queuing theory is the study of the distribution of the arrival time and service time of the customers. We can use queueing theory to optimize queueing systems to meet the quantity of service with minimal cost. Kendall's notation is the standard system used to describe and classify queuing models. A queue is described in shorthand notation by X/Y/Z/A/B/C. The meaning of them is explained below:

- X: distribution of arrival time
- Y: distribution of service time

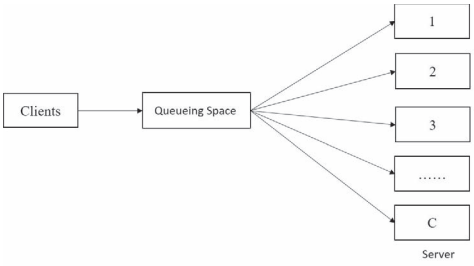


Fig. 1: Queueing model of M/M/C.

- Z: the number of parallel servers
- A: the capacity of the system. It refers to the maximum number of customers allowed in the system including those in service.
- B: the calling population. The size of the population from which the customers come. If this number is omitted, the population is assumed to be unlimited, or infinite.
- C: the service disciplines.

Here we give an example of the queueing model. Suppose there are C system service servers, where each server is independent. Customer's arrival time interval follows exponential distribution with parameter λ and each server's service time follows exponential distribution with parameter μ. We call such a system with multi-service windows of queuing model noted by M/M/C. Fig. 1 shows this example. Based on queueing theory [12], we have the following results.

1) Traffic intensity of the system

$$\rho = \frac{\lambda}{c\mu} \tag{1}$$

2) The idle probability of the system

$$P_0 = [\sum_{n=0}^{c-1} \frac{(c\rho)^n}{n!} + \frac{(c\rho)^c}{c!(1-\rho)}]^{-1} \tag{2}$$

3) The blocking probability

$$P_b = \frac{c^c}{c!} P^k P_0 \tag{3}$$

4) Average number of customers in the Queue

$$L_q = \frac{(c\rho)^c \rho}{c!(1-\rho)^2} P_0 \tag{4}$$

5) Average number of customers in the system

$$L_s = \frac{(c\rho)^c \rho}{c!(1-\rho)^2} P_0 + c\rho \tag{5}$$

6) Expected waiting time in queue

$$W_q = \frac{(c\rho)^c}{c!(c\rho)(1-\rho)^2} P_0 \tag{6}$$

7) Expected waiting time in system

$$W_s = \frac{(c\rho)^c}{c!(c\rho)(1-\rho)^2} P_0 + \frac{1}{\mu} \tag{7}$$

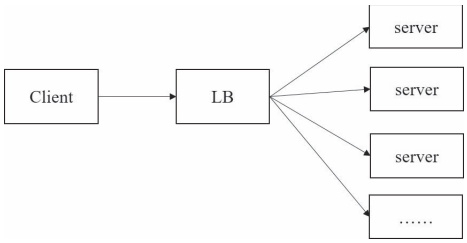


Fig. 2: Simple structure of the system.

Algorithm 1 Resource Allocation Algorithm

Input: $\mathcal{H}_t = (\lambda_t, c_t, \mu_t, V_{i,t}, \theta)$
Output: $\mathcal{V}_{min}$

compute $\rho_{t+1}, P_{0,t+1}$ and $W_{s,t+1}$ by Equs. (8)(9)(11)
$i \leftarrow 1$
while $f(i) < f(i+1)$ **do**
    $i \leftarrow i + 1$
end while
return i

Algorithm 2 The Virtual Machines Scale-Up Algorithm

Input: $V_{i,t}, c_t, V_{t+1}, A_i, B$
Output: The operation to scale-up the virtual machines

$V'_{t+1} \leftarrow V_{t+1} - \sum_{i=1}^{c_t} V_{i,t}$
$i \leftarrow 1$
while $V'_{t+1} > 0$ and $i <= c_t$ **do**
    if $V'_{t+1} > min(A_i, B - V_{i,t})$ **then**
        scale-up i-th VM with resource $\min(A_i, B - V_{i,t})$
        $V'_{t+1} \leftarrow V'_{t+1} - min(A_i, V_{i,t})$
    else
        scale-up i-th VM with resource V'_{t+1}
        $V'_{t+1} \leftarrow 0$
    end if
    $i \leftarrow i + 1$
end while
if $V'_{t+1} > 0$ **then**
    start $\lfloor V'_{t+1}/B \rfloor$ VMs with resource B
    start a VM with resource $V'_{t+1} - \lfloor V'_{t+1}/B \rfloor * B$
end if

B. Problem Definition

The system we consider is shown in Fig. 2. It has a load balancer (LB) and C servers where each server is deployed in a VM. When a client accesses to the web service, it is queued at the LB until one server can serve the client. We model the system using a M/M/C queue. In the model, the clients arrivals follow the Poisson distribution with parameter λ.

We denote that average arrival speed is λ and the number of servers is n, the average service speed of the i-th server is μ_i, and the resource of i-th server is V_i.

In order to facilitate the calculation, we assume that V_i is an integer, and μ_i is proportional to V_i. We definite the state of the system in time interval t is

$$H_t = (\lambda_t, n_t, \mu_{i,t}, V_{i,t}, \theta), i = 1, 2, 3, \cdots, n$$

Then the total service speed of the system is

$$\mu_t = \sum_{i=1}^{n} \mu_{i,t},$$

and the total resource of the system is

$$V_t = \sum_{i=1}^{n} V_{i,t}.$$

We assume the average arrival rate of the clients will not change in a short time. If we set a small time interval, we can consider the average arrival rate in time interval t is equal to $t+1$, that is $\lambda_{t+1} = \lambda_t$.

According to the queueing theory results in the last section, we can get the performance metrics of our system in time interval t, including the idle probability of the system $P_0(V_{t+1})$, average number of customers in the Queue $L_q(V_{t+1})$, average number of customers in the system $L_s(V_{t+1})$, expected waiting time in queue $W_q(V_{t+1})$, and expected waiting time in system $W_s(V_{t+1})$.

Problem 1: (**Elastic resource allocation problem**) Given the state of system in time interval t $H_t = (\lambda_t, c_t, \mu_{i,t}, V_{i,t}, \theta)$, how to minimize the demand of resources of the system and expected queueing time in time interval $t+1$,

$$\min f(V_{t+1}) = V_{t+1} + \alpha \cdot W_s(V_{t+1})$$

IV. Solution

A. Resource Allocation Algorithm

The main idea of the resource allocation algorithm is to compute the optimal demand of resources V_{t+1} in time interval $t+1$ according the state of the system H_t in time interval t.

According to Equs. (1)(2)(6)(7) in Section III, we can get the traffic intensity of the system

$$\rho_{t+1} = \frac{\lambda_{t+1}}{V_{t+1}\mu_{t+1}}, \tag{8}$$

and the idle probability of the system

$$P_{0,t+1} = [\sum_{n=0}^{c_{t+1}-1} \frac{(c_{t+1}\rho_{t+1})^n}{n!} + \frac{(c_{t+1}\rho_{t+1})^{c_{t+1}}}{c_{t+1}!(1-\rho_{t+1})}]^{-1} \tag{9}$$

Then we can get the average waiting time in the queue

$$W_{q,t+1} = \frac{(c_{t+1}\rho_{t+1})^{c_{t+1}}}{c_{t+1}!(c_{t+1}\rho_{t+1})(1-\rho_{t+1})^2} P_{0,t+1} \tag{10}$$

and the average waiting time in the system

$$W_{s,t+1} = \frac{(c_{t+1}\rho_{t+1})^{c_{t+1}}}{c_{t+1}!(c_{t+1}\rho_{t+1})(1-\rho_{t+1})^2} P_{0,t+1} + \frac{1}{\mu_{t+1}} \tag{11}$$

Then, the objective function is

$$f(V_{t+1}) = V_{t+1} + \alpha \cdot W_s(V_{t+1}). \tag{12}$$

It is not hard to see that $f(V_{t+1})$ is a discrete function with U-shaped trend. So we can use a simple circulation to get the optimal resource as shown in Alg. 1.

Algorithm 3 The Virtual Machines Scale-Down Algorithm

Input: $V_{i,t}, c_t, V'_{t+1}$
Output: The operation to scale-down the virtual machines

Create a two-dimensional array $A[c_t + 1][V'_{t+1} + 1]$
Initial the first row and first column of A to 0
for $(i = 1; i <= c_t; i = i + 1)$ **do**
    for $(j = 1; j <= V'_{t+1}; j = j + 1)$ **do**
        if $A[i-1][j] > (A[i-1][j - V_{i,t}] + V_{i,t})$ **then**
            $A[i][j] = A[i-1][j]$
        else
            $A[i][j] = A[i-1][j - V_{i,t}] + V_{i,t}$
        end if
    end for
end for
Create an array $B[c_t + 1]$
$w = V'_{t+1}$
for $(i = c_t; i >= 2; i = i - 1)$ **do**
    if $A[i][w] == A[i-1][w]$ **then**
        $B[i] = 0$
    else
        $B[i] = 1$
    end if
end for
if $A[i][w] == 0$ **then**
    $B[1] = 0$
else
    $B[1] = 1$
end if
return B

B. Auto-scaling virtual machines

We now give an elastic auto-scaling policy of virtual machines to decide whether to scale-up or scale-down the resource and how much to scale.

The basic idea behind our algorithm is as follows: according to Alg. 1, we can get the optimal resources V_{t+1}. Then we compare V_{t+1} with the actual resources V_t in time interval t . If $V_{t+1} > V_t$, we need to scale-up the resources; If $V_{t+1} < V_t$,we need to scale-down the resources.

1) Scale-up the virtual machines: Denote the remaining resources of each virtual machine's host by A_i. If there are k virtual machines in the same host, set the A_i to the average remaining resources. Denote B to the max resource that a VM supports.

As shown in Alg. 2, we use heuristic algorithms to get which VM to scale-up and how much to scale up.

2) Scale-down the virtual machines: If $V_{t+1} < V_t$, we need to scale-down VMs. Because the action that vertically scales down a VM may cause a failure, we only consider horizontal scale-down. Then, the scale-down problem becomes a 0-1 Knapsack problem, so we use dynamic programming (DP) to solve it. With DP, we need to choose some VMs to close and the rest of the resources must be larger than $V_t - V_{t+1}$.

Denote by V'_{t+1} the resources that need to be scaled down, so $V'_{t+1} = V_t - V_{t+1}$.

Let $f(i, j)$ represent the profit that i VMs can get while $f(i, j)$ must be smaller than j.

As shown in Alg. 3, we first use DP to know which virtual machines should be closed. At the end of the algorithm, we can get an array B. If $B[i]$ is 1, we should close the $i-th$ VM. According to array B, we can optimally scale-down VMs.

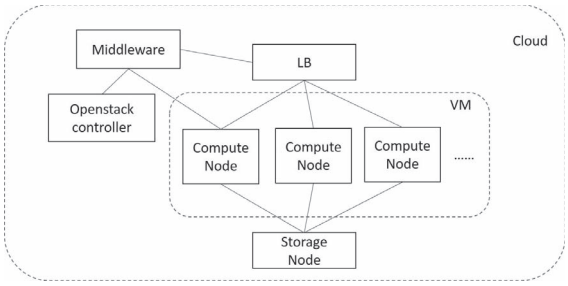
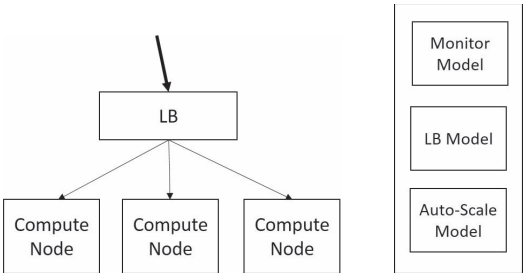


Fig. 3: System architecture.



(a) LB Model (b) Middleware models

Fig. 4: System details

V. System Overview

Elastic web application middleware is a system between user and cloud platform. It can help user to automatically deploy VMs in the clouds. By monitoring these VMs (including CPU and memory usage), it self-adaptively adjust the allocation of resources for VMs, including vertically scale and horizontally scale. Horizontal scaling is to increase or reduce the number of VMs, and vertical scaling is the resize of the VM (change the CPU or memory of the VM). For example, when the VM overall load is very high, the middleware will perform a vertical scaling. If resources still can not meet the needs, and then start a horizontal scaling. It can dynamically use cloud resources based on resource demands in order to improve resources utilization and meanwhile reduce deployment costs.

As shown in Fig. 3, the system is composed of several parts, including Middleware, Openstack Controller, Load Balancer(LB), Compute Node, and Storage Node.

The Openstack Controller offers some Rest APIs that we can get launch or terminate or resize some VMs in the openstack.

The Load Balancer, which is shown in Fig. 4a, can forward the request of clients to different servers to balance the load of the servers.

As shown in Fig. 4b, the proposed middleware is composed of three models, including minitor model, LB model, and auto-

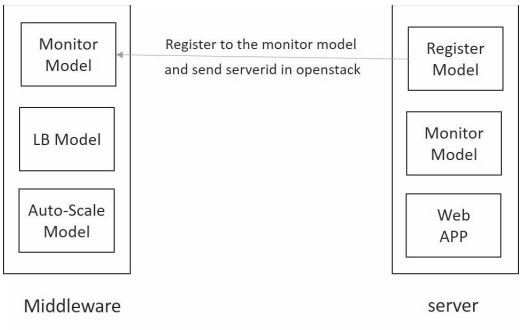


Fig. 5: Monitor Model: registration.

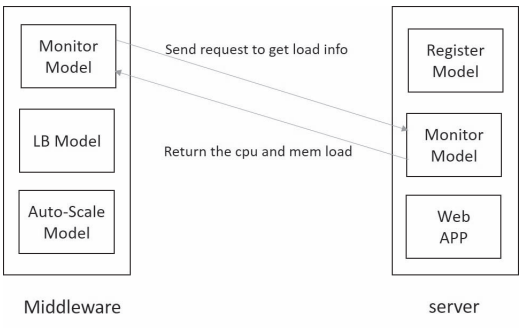


Fig. 6: Monitor Model: request.

scale model. When the middleare in the monitor model, it will monitor the state of the VMs, including the CPU and MEM usage. And the monitor component in compute nodes will assist the former to complete the monitor process.

As shown in Fig. 5, when a VM is launched, it will register to the monitor component with its IP address and serverid of the itself. As shown in Fig. 6, when the monitor component sends request to the monitor component in some compute node, it will return its CPU and MEM load statistics to the former. As shown in Fig. 7, the LB model will inform load balancer to change the weight of VMs as the load varies. And when a VM is terminated, it will inform load balancer to remove the VM in the server list. As shown in Fig. 8,the auto-scale model will scale-up or scale-down the VMs in the cloud.

VI. Performance Evaluation

In this section, we conduct simulations to evaluate our design under different parameters. We use Openstack as the platform to run experiments to evaluate the effectiveness of the solutions.

A. Setup

Denote by λ the average number of resource requests that arrive at our system in unit time, and λ is set to 300 in our experiments. Denote by μ the average number of resource requests that one unit of resource that process, and μ is set to 50 throughout our experiments. The request arrivals follow Possion distribution in our experiments, as shown in Fig. 9a.

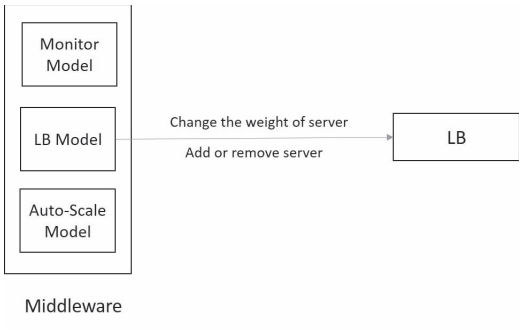


Fig. 7: Load Balance Model.

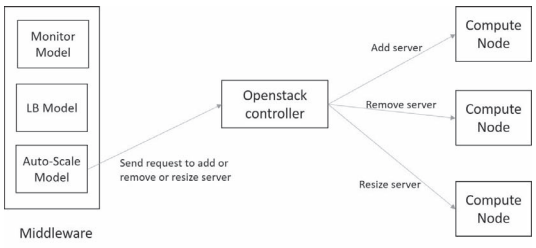


Fig. 8: Auto-scaling.

B. Results

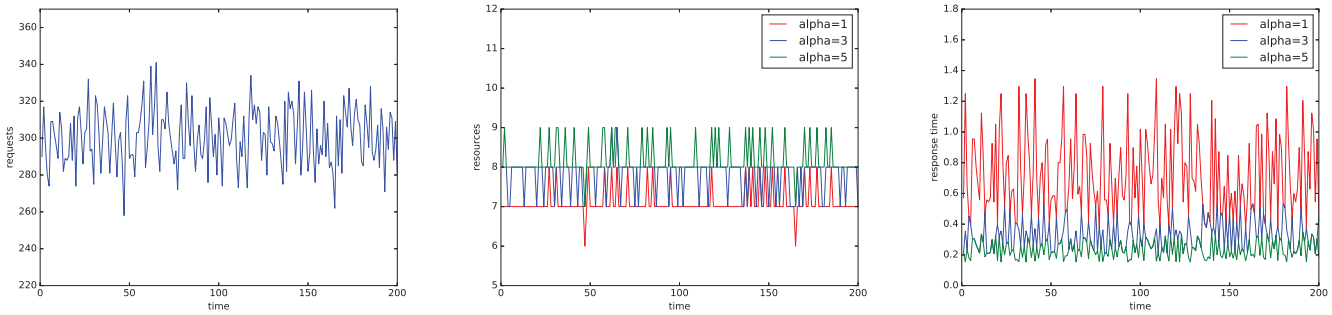
First of all, we are interested in the effects of α in our objective function on the amount of required resources and response time. The results are shown in Figs. 9b and 9c.

The parameter α is larger, the response time in our objective function is more important. Therefore, the system gets better performance and the demand of resources is larger. By previous methods, we may allocate the resources corresponding to the peak load but the peak load may maintain a very short time. If we use auto-scaling method, we can adjust the resources with the change of the requests. As a result, we do not need to always maintain the maximum resources and we can save a large plenty of resources.

To evaluate the performance of the Auto-scaling Algorithm, we conduct some other experiments. We use multiple threads to simulate the virtual machines. Every virtual machine corresponds to a thread. Every thread will maintain a value which represents the number of requests in the virtual machine. Every unit time this value will be added a number which represent the number of requests distributed by the Load Balancer and will be reduced which represent the number of requests served by the virtual machine.

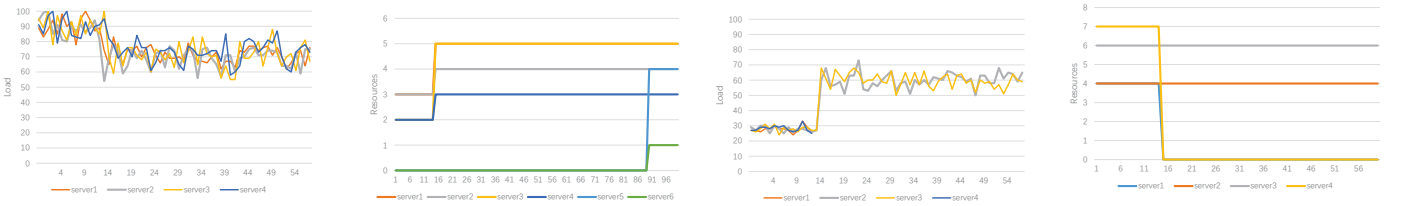
We define the workload to be the actual requests served by the virtual machine divided by the capacity of the virtual machines in unit time. We assume that a virtual machine can process 50 requests in unit time.

Firstly, we simulate the scaling-up algorithm. We suppose that the average number of requests λ is 500 and there are four virtual machines whose resources are 4, 3, 2, and 2, respectively. As shown in Fig. 10a, the load of virtual machine is very large and service quality is very poor. According to the resource allocation algorithm and scaled-up algorithm, the



(a) Request arrivals follow poisson distribution (b) Resource required under different α's (c) Response time

Fig. 9: Effect of system parameters



(a) Load variation in scaling up (b) Resources variation in scaling up (c) Load variation in scaling down (d) Resources variation in scaling down

Fig. 10: Performance of auto scaling algorithms

resources of the four virtual machines change to 5, 5, 4, and 2, respectively, then, the load of the virtual machine become lower, as shown in Fig. 10b.

Then we simulate the scaling-down algorithm. We assume that the average number of requests in unit time is 300 and there are four virtual machines whose resources are 4, 4, 6, and 7, respectively. As shown in Fig. 10c, the load of virtual machine is very low and we do not need so much resources. According to the resource allocation algorithm and scaled-down algorithm, the first and fourth virtual machines should be terminated, as shown in Fig. 10d. Then the load of the virtual machine become larger and resource utilization become larger.

In summary, our experimental results confirm that the proposed model and scaling up/down algorithms are effective and efficient in terms of both response time and resource utilization.

VII. Conclusions

We studied the problem of resource allocation in cloud computing platform with the aim of reducing the usage of resource. To solve the problem, we predict the best the resources of the application by queueing-theory. Then we design an virtual machine auto-scaling mechanism which automatically adjust the resources of the application according to the best resources predicted by the queueing-theory. Compared with traditional method that fixedly deploy resources by the peak load, our method can effectively improve the resource utilization and guarantee the quality of service.

References

[1] S. Zhang, Z. Qian, J. Wu, and S. Lu, "An opportunistic resource sharing and topology-aware mapping framework for virtual networks," in *INFOCOM, 2012 Proceedings IEEE*, pp. 2408–2416, March 2012.

[2] S. Zhang, Z. Qian, J. Wu, S. Lu, and L. Epstein, "Virtual network embedding with opportunistic resource sharing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, pp. 816–827, March 2014.

[3] S. Zhang, Z. Qian, Z. Luo, J. Wu, and S. Lu, "Burstiness-aware resource reservation for server consolidation in computing clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, pp. 964–977, April 2016.

[4] E. Barrett, E. Howley, and J. Duggan, "Applying reinforcement learning towards automating resource allocation and application scalability in the cloud," *Concurrency and Computation: Practice and Experience*, vol. 25, no. 12, pp. 1656–1674, 2013.

[5] G. Tesauro, N. K. Jong, R. Das, and M. N. Bennani, "A hybrid reinforcement learning approach to autonomic resource allocation," in *2006 IEEE International Conference on Autonomic Computing*, pp. 65–73, IEEE, 2006.

[6] 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.

[7] T. Lorido-Botrán, J. Miguel-Alonso, and J. A. Lozano, "Auto-scaling techniques for elastic applications in cloud environments," *Department of Computer Architecture and Technology, University of Basque Country, Tech. Rep. EHU-KAT-IK-09-12*, 2012.

[8] H. C. Lim, S. Babu, and J. S. Chase, "Automated control for elastic storage," in *Proceedings of the 7th international conference on Autonomic computing*, pp. 1–10, ACM, 2010.

[9] T. Patikirikorala, A. Colman, J. Han, and L. Wang, "A multi-model framework to implement self-managing control systems for qos management," in *Proceedings of the 6th international symposium on software engineering for adaptive and self-managing systems*, pp. 218–227, ACM, 2011.

[10] A. Ali-Eldin, J. Tordsson, and E. Elmroth, "An adaptive hybrid elasticity controller for cloud infrastructures," in *2012 IEEE Network Operations and Management Symposium*, pp. 204–212, IEEE, 2012.

[11] J. Jiang, *Optimised auto-scaling for cloud-based web service*. PhD thesis, University of Technology Sydney, 2015.

[12] R. B. Cooper, *Introduction to queueing theory*. North Holland,, 1981.